

CoRE
Internet-Draft
Intended status: Standards Track
Expires: August 24, 2017

K. Kuladinithi, Ed.
ComNets, Hamburg University of Technology
M. Becker
Tridonic GmbH & Co KG
K. Li
Alibaba Group
T. Poetsch
New York University Abu Dhabi
February 20, 2017

Transport of CoAP over SMS
draft-becker-core-coap-sms-gprs-06

Abstract

Short Message Service (SMS) of mobile cellular networks is frequently used in Machine-To-Machine (M2M) communications, such as for telematic devices. The service offers small packet sizes and high delays just as other typical low-power and lossy networks (LLNs), i.e. 6LoWPANs. The design of the Constrained Application Protocol (CoAP, RFC7252), that took the limitations of LLNs into account, is thus also applicable to other transports. The adaptation of CoAP to SMS transport mechanisms is described in this document.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <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 August 24, 2017.

Copyright Notice

Copyright (c) 2017 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
1.1. Motivation	3
1.2. Terminology	3
1.3. Requirements Language	4
2. Scenarios	4
2.1. MO-MT Scenarios	4
2.2. MT Scenarios	4
2.3. MO Scenarios	5
3. Message Exchanges	6
3.1. Message Exchange for SMS in a Cellular-To-Cellular Mobile-Originated and Mobile-Terminated Scenario	6
4. Encoding Schemes of CoAP for SMS transport	7
5. Message Size Implementation Considerations	7
6. Addressing	8
7. Options	8
7.1. New Options for mixed IP operation.	8
8. URI Scheme	9
9. Transmission Parameters	9
10. Multicast	9
11. Security Considerations	9
12. IANA Considerations	10
12.1. CoAP Option Number	10
12.2. URI Scheme Registration	10
13. References	10
13.1. Normative References	10
13.2. Informative References	11
Appendix A. SMS encoding	12
A.1. ASCII-optimized SMS encoding	12
Appendix B. Changelog	16
Acknowledgements	17
Contributors	17
Authors' Addresses	17

1. Introduction

This specification details the usage of the Constrained Application Protocol on the Short Message Service (SMS) of mobile cellular networks.

1.1. Motivation

In some M2M environments, internet connectivity is not supported by the constrained end-points, but a cellular network connection is supported instead. Internet connectivity might also be switched off for power saving reasons or the cellular coverage does not allow for Internet connectivity. In these situations, SMS will be supported, instead of UDP/IP over General Packet Radio Service (GPRS), High Speed Packet Access (HSPA) or Long Term Evolution (LTE) networks.

In 3GPP, SMS is identified as the transport protocol for small data transmissions (See [ts23_888] for Key Issue on Machine Type Communication (MTC) Device Trigger and the proposed solutions in Sections 6.2, 6.42, 6.44, 6.48, 6.52, 6.60, and 6.61). In [ts23_682] 'Architecture Enhancements to facilitate communications with Packet Data Networks and Applications' SMS is at the moment the only Trigger Delivery (Trigger Delivery using T4).

M2M protocols using SMS, e.g. for telematics, are using mostly various diverse proprietary and closed binary protocols with limited publicly available documentation at the moment.

In Open Mobile Alliance (OMA) LightweightM2M technical specification [oma_lightweightm2m_ts], SMS is identified as an alternative transport for CoAP messages.

1.2. Terminology

This document uses the following terminology:

CoAP Server and Client

The terms CoAP Server and CoAP Client are used synonymously to Server and Client as specified in the terminology section of [RFC7252].

Mobile Station (MS)

A Mobile Station includes all required user equipment and software that is needed for communication with a mobile network. As defined in [etsi_ts101_748].

1.3. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Scenarios

Several scenarios are presented first for M2M communications with CoAP over SMS. First Mobile-Originating Mobile-Terminating (MO-MT) scenarios are presented, where both CoAP endpoints are in devices in a cellular network. Next, Mobile-Terminating (MT) scenarios are detailed, where only the CoAP server is in a cellular network. Finally, Mobile-Originating (MO) scenarios where the CoAP client is in the cellular network.

2.1. MO-MT Scenarios

Two mobile cellular terminals communicate by exchanging a CoAP Request and Response embedded into short message protocol data units (PDUs) (depicted in Figure 1). Both terminals are connected via a Short Message Service Centre (SMS-C).

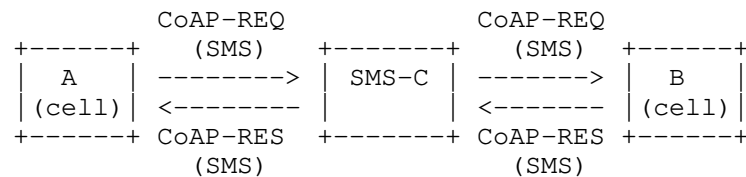


Figure 1: Cellular and Cellular Communication (only SMS-based)

2.2. MT Scenarios

An IP host and a mobile cellular terminal communicate by exchanging CoAP Request and Response. The IP host uses protocols offered by the SMS-C (e.g. Short Message Peer-to-Peer (SMPP [smpp]), Computer Interface to Message Distribution (CIMD [cimd]), Universal Computer Protocol/External Machine Interface (UCP/EMI [ucp])) to submit a short message for delivery, which contains the CoAP Request (depicted in Figure 2).

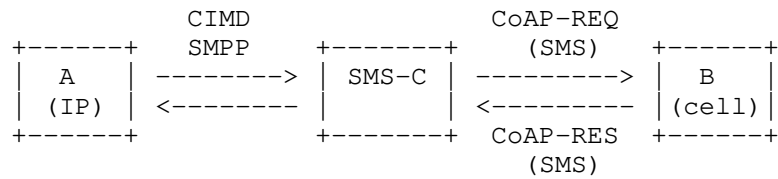


Figure 2: IP and Cellular Communication

There are service providers that offer SMS delivery and notification using an HTTP/REST interface (depicted in Figure 3).

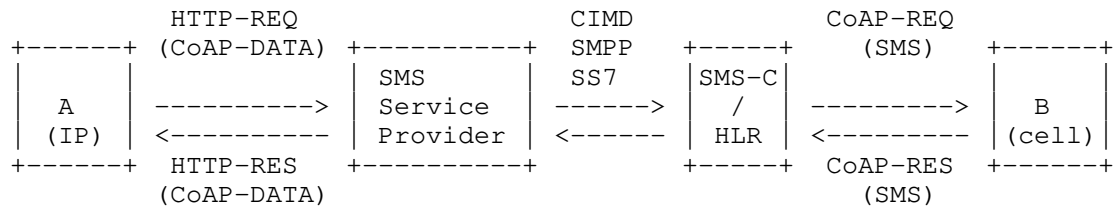


Figure 3: IP and Cellular Communication (using an SMS Service Provider)

2.3. MO Scenarios

A mobile cellular terminal and an IP host communicate by exchanging CoAP Request and Response. The mobile cellular terminal sends a CoAP Request in a short message, which is in turn forwarded by the SMS-C (e.g. with Short Message Peer-to-Peer (SMPP [smpp]), Computer Interface to Message Distribution (CIMD [cimd]), Universal Computer Protocol/External Machine Interface (UCP/EMI [ucp])) as depicted in Figure 4). This scenario can be a fall-back for mobile-originating communication, when IP connectivity cannot be setup (e.g. due to missing coverage).

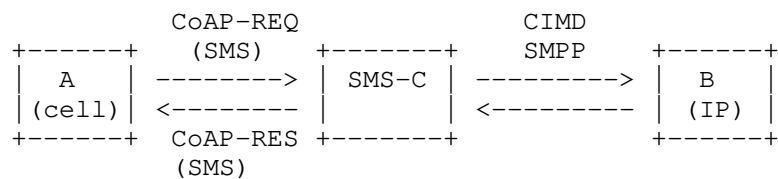


Figure 4: Cellular and IP Communication

There are service providers offering SMS delivery and notification using an HTTP/REST interface (depicted in Figure 5).

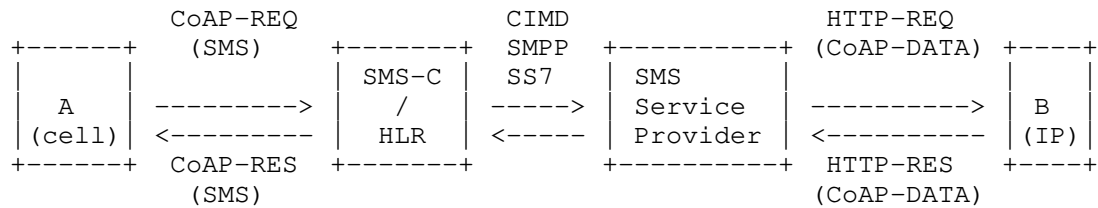


Figure 5: IP and Cellular Communication (using an SMS Service Provider)

3. Message Exchanges

3.1. Message Exchange for SMS in a Cellular-To-Cellular Mobile-Originated and Mobile-Terminated Scenario

The CoAP Client works as a Mobile Station to send the short message, and the CoAP Server works as another Mobile Station to receive the short message. All short messages are stored and forwarded by the Service Center. The message exchange between the CoAP Client and the CoAP Server is depicted in the figure below:

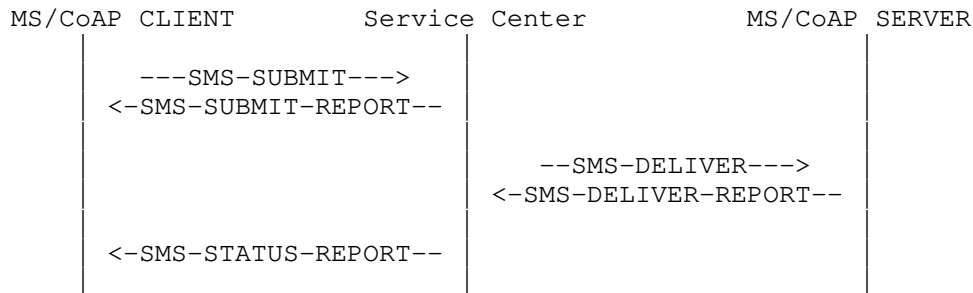


Figure 6: CoAP Messages over SMS

Note that the message exchange is just for one request message from CoAP Client and CoAP Server. It includes the following steps:

Step 1: The CoAP Client sends a CoAP request in a SMS-SUBMIT message to the Service Center. The CoAP Server address is specified as TP-Destination-Address (see [ts23_040]).

Step 2: The Service Center returns a SMS-SUBMIT-REPORT message to the CoAP Client.

Step 3: The Service Center stores the received SMS message and forwards it to the CoAP Server, using an SMS-DELIVER message. The

CoAP Client address is specified as a TP Originating Address (see [ts23_040]).

Step 4: The CoAP Server returns an SMS-DELIVER-REPORT message to the Service Center.

Step 5: The Service Center returns the SMS-STATUS-REPORT message to the CoAP Client to indicate the SMS delivery status, if required by the CoAP Client.

Note that the SMS-STATUS-REPORT message just indicates the transport layer SMS delivery status and has no relationship with the confirmable message or non-confirmable message. If the CoAP Client has sent a confirmable message, the CoAP Server MUST use a separate SMS message to transmit the ACK.

4. Encoding Schemes of CoAP for SMS transport

Short messages can be encoded by using various alphabets: GSM 7 bit default alphabet ([ts23_038]), 8 bit data alphabet, and 16 bit UCS2 data alphabet ([iso_ucs2]). These encodings lead to message sizes of 160, 140, and 70 characters, respectively. Whereas the support of 7 bit encoding is mandatory on a MS, the two other encodings are dependent on the language that needs to be encoded, e.g. UCS2 for Arabic, Chinese, Japanese, etc. Furthermore, the supported encoding highly depends on the implementations of the MS itself.

According to [ts23_038], GSM 7 bit encoding shall be supported by all MSs offering SMS services. Since not all MSs support 8 bit short message encoding, the preferred encoding scheme for CoAP messages over SMS is therefore 7 bit, e.g. Base64 ([RFC4648]) or SMS encoding in Appendix A.1.

More considerations about SMS encoding can be found in Appendix A.

5. Message Size Implementation Considerations

By using 7 bit encoding, a maximum length of 160 characters is allowed in one short message [ts23_038]. Consequently, the maximum length for a CoAP message results in 140 bytes. $160 \text{ characters} = (140 \text{ bytes} * 8) / 7$.

Possible options for larger CoAP messages are:

Concatenated short messages

Most MSs are able to send concatenation short messages in order to transmit longer messages. The total length of a concatenated short message can consist of up to 255 single messages and result

in total length of 39015 7 bit characters or 34170 bytes.
 Resulting from this, the maximum length of each individual message reduces to 153 (160 - 7) characters (133 bytes).

CoAP block-wise transfer

According to [RFC7959], the Block Size (SZX) of block-wise transfer in CoAP is represented as a three-bit unsigned integer. Thus, the possible block sizes are to the power of two. (Block size = $2^{(SZX + 4)}$). Due to the limitations of 160 characters (140 bytes) for one short message, the maximum value of SZX is 3 (Block size = 128 byte).

However, it is RECOMMENDED that SMS is not used to transfer very large resource data using block-wise transfer.

6. Addressing

For SMS in cellular networks, the CoAP endpoints have to work with a SIM (Subscriber Identity Module) card and have to be addressed by the MSISDN (Mobile Station ISDN (MSISDN) number).

To allow the CoAP client to detect that the short message contains a CoAP message, the TP-DATA-Coding-Scheme SHOULD be included.

7. Options

7.1. New Options for mixed IP operation.

In case a CoAP Server has more than one network interface, e.g. SMS and IP, the CoAP Client might want the server to send the response via an alternative transport, i.e. to its alternative address. However, that implies that the initiating CoAP Client is aware of the presence of the alternative interface. For this reason the new options Response-To-Uri-Host and Response-To-Uri-Port are proposed.

No.	C	U	N	R	Name	Format	Length	Default
TBD					Response-To-Uri-Host	string	1-270 B	(none)
TBD					Response-To-Uri-Port	uint	0-2 B	5683

Table 1: New CoAP Option Numbers

If the Response-To-Uri-Host is present in the request, server MUST send the response to the indicated URI address, instead of the client's original request URI.

The options SHOULD NOT be used in the response.

The options MUST NOT occur more than once.

8. URI Scheme

The coap:// scheme defines that a CoAP server is reachable over UDP/IP. Hence, a new URI scheme is needed for CoAP servers which are reachable over SMS.

As proposed in [I-D.silverajan-core-coap-alternative-transport], the transport information is expressed as part of the URI scheme component. This is performed by minting new schemes for SMS transport using the form "coap+sms", where the name of the transport is clearly and unambiguously described. The endpoint identifier, path and query components together with each scheme name would be used to uniquely identify each resource.

Example of such URI :

o coap+sms://0015105550101/sensors/temperature

In the URI, 0015105550101 is a telephone subscriber number.

9. Transmission Parameters

It is RECOMMENDED to configure the RESPONSE_TIMEOUT variable for a higher duration than specified in [RFC7252] for the applications described here. The actual value SHOULD be chosen based on experience with SMS.

10. Multicast

Multicast is not possible with SMS transports.

11. Security Considerations

It is possible that a malicious CoAP Client sends repeated requests, and it may cost money for the CoAP Server to use SMS to send back associated responses. To avoid this situation, the CoAP Server implementation can authenticate the CoAP Client before responding to the requests. For example, the CoAP Server can maintain an MSISDN white list. Only the MSISDN specified in the white list will be

allowed to send requests. The requests from others will be ignored or rejected.

12. IANA Considerations

12.1. CoAP Option Number

The IANA is requested to add the following option number entries to the CoAP Option Number Registry:

Number	Name	Reference
TBD	Response-To-Uri-Host	Section 2 of this document
TBD	Response-To-Uri-Port	Section 2 of this document

12.2. URI Scheme Registration

According to [I-D.silverajan-core-coap-alternative-transport] this document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+sms". The registration request complies with [RFC7595].

13. References

13.1. Normative References

- [etsi_ts101_748] ETSI, "Technical Report: Digital cellular telecommunications system; Abbreviations and acronyms (GSM 01.04 version 8.0.0 release 1999)", 2000.
- [iso_ucs2] ISO, "ISO/IEC10646: "Universal Multiple-Octet Coded Character Set (UCS)"; UCS2, 16 bit coding.", 2000.
- [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>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<http://www.rfc-editor.org/info/rfc4648>>.

- [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>>.
- [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, <<http://www.rfc-editor.org/info/rfc7595>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<http://www.rfc-editor.org/info/rfc7959>>.
- [ts23_038] ETSI 3GPP, "Technical Specification: Alphabets and language-specific information (3GPP TS 23.038 version 11.0.0 Release 11)", 2012.

13.2. Informative References

- [cimd] Nokia, "CIMD Interface Specification (SMSCDOC8000.00, Nokia SMS Center 8.0)", 2005.
- [I-D.silverajan-core-coap-alternative-transport] Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transport-09 (work in progress), December 2015.
- [oma_lightweightm2m_ts] OMA, "Lightweight Machine to Machine Technical Specification", 2013.
- [RFC1924] Elz, R., "A Compact Representation of IPv6 Addresses", RFC 1924, DOI 10.17487/RFC1924, April 1996, <<http://www.rfc-editor.org/info/rfc1924>>.
- [smpp] SMPP Developers Forum, "Short Message Peer to Peer Protocol Specification v3.4 Issue 1.2", 1999.
- [ts23_040] 3GPP, "Technical realization of the Short Message Service (SMS)", 3GPP-23.040 a00, March 2011.

- [ts23_682] ETSI 3GPP, "Technical Specification Group Services and System Aspects; Architecture Enhancements to facilitate communications with Packet Data Networks and Applications; (Release 11)", 2012.
- [ts23_888] ETSI 3GPP, "Technical Specification Group Services and System Aspects; System Improvements for Machine-Type Communications; (3GPP TR 23.888 version 1.6.0, Release 11)", 2011.
- [ucp] Vodafone, "Short Message Service Centre (SMSC) External Machine Interface (EMI) Description Version 4.3d", 2011.

Appendix A. SMS encoding

For use in SMS applications, CoAP messages can be transferred using SMS binary mode. However, there is operational experience showing that some environments cannot successfully send a binary mode SMS.

For transferring SMS in character mode (7-bit characters), base64-encoding [RFC4648] is an obvious choice. 3 bytes of message (24 bits) turn into 4 characters, which consume 28 bits. The overall overhead is approximately 17 %; the maximum message size is 120 bytes (160 SMS characters).

If a more compact encoding is desired, base85 encoding could be employed (however, probably not the version defined in [RFC1924] -- instead, the version used in tools such as btoa and PDF should be chosen). However, this requires division operations. Also, the base85 character set includes several characters that cannot be transferred in a single 7-bit unit in SMS and/or are known to cause operational problems. A modified base85 character set can be defined to solve the latter problem. 4 bytes of message (32 bits) turn into 5 characters, which consume 35 bits. The overall overhead is approximately 9.3 %; the resulting maximum message size is 128 bytes (160 SMS characters).

Base64 and base85 do not make use of the fact that much CoAP data will be ASCII-based. Therefore, we define the following ASCII-optimized SMS encoding.

A.1. ASCII-optimized SMS encoding

Not all 128 theoretically possible SMS characters are operationally free of problems. We therefore define:

Shunned code characters: @ sign, as it maps to 0x00

LF and CR signs (0x0A, 0x0D)

uppercase C cedilla (0x09), as it is often mistranslated in gateways

ESC (0x1B), as it is used in certain character combinations only

Some ASCII characters cannot be transferred in the base SMS character set, as their code positions are taken by non-ASCII characters. These are simply encoded with their ASCII code positions, e.g., an underscore becomes a section mark (even though underscore has a different code position in the SMS character set).

Equivalently translated input bytes: \$, @, [, \,], ^, _, ` , {, |, }, ~, DEL

In other words, bytes 0x20 to 0x7F are encoded into the same code positions in the 7-bit character set.

Out of the remaining code characters, the following SMS characters are available for encoding:

Non-equivalently translated (NET) code characters: 0x01 to 0x08, (8 characters)

0x0B, 0x0C, (2 characters)

0x0E to 0x1A, (13 characters)

0x1C to 0x1F, (4 characters)

Of the 27 NET code characters, 18 are taken as prefix characters (see below), and 8 are defined as directly translated characters:

Directly translated bytes: Equivalently translated input bytes are represented as themselves

0x00 to 0x07 are represented as 0x01 to 0x08

This leaves 0x08 to 0x1F and 0x80 to 0xFF. Of these, the bytes 0x80 to 0x87 and 0xA0 to 0xFF are represented as the bytes 0x00 to 0x07 (represented by characters 0x01 to 0x08) and 0x20 to 0x7F, with a prefix of 1 (see below). The characters 0x08 to 0x1F are represented as the characters 0x28 to 0x3F with a prefix of 2 (see below). The characters 0x88 to 0x9F are represented as the characters 0x48 to 0x5F with a prefix of 2 (see below). (Characters 0x01 to 0x08, 0x20

to 0x27, 0x40 to 0x47, and 0x60 to 0x7f with a prefix of 2 are reserved for future extensions, which could be used for some backreferencing or run-length compression.)

Bytes that do not need a prefix (directly translated bytes) are sent as is. Any byte that does need a prefix (i.e., 1 or 2) is preceded by a prefix character, which provides a prefix for this and the following two bytes as follows:

char	pxf	.	char	pxf
0x0B	100	.	0x15	200
0x0C	101	.	0x16	201
0x0E	102	.	0x17	202
0x0F	110	.	0x18	210
0x10	111	.	0x19	211
0x11	112	.	0x1A	212
0x12	120	.	0x1C	220
0x13	121	.	0x1D	221
0x14	122	.	0x1E	222

Table 2: SMS prefix character assignment

(This leaves one non-shunned character, 0x1F, for future extension.)

The coding overhead of this encoding for random bytes is similar to Base85, without the need for a division/multiplication. For bytes that are mostly ASCII characters, the overhead can easily become negative. (Conversely, for bytes that for some reason are more likely to be non-ASCII than in a random sequence of bytes, the overhead becomes greater.)

So, for instance, for the CoAP message in Figure 7:

ver	tt	code	mid	
1	ack	2.05	17033	
content_type		40		
token		sometok		

3c 2f 3e 3b 74 69 74 6c 65 3d 22 47 65 6e 65 72	</>;title="Gener
61 6c 20 49 6e 66 6f 22 3b 63 74 3d 30 2c 3c 2f	al Info";ct=0,</
74 69 6d 65 3e 3b 69 66 3d 22 63 6c 6f 63 6b 22	time>;if="clock"
3b 72 74 3d 22 54 69 63 6b 73 22 3b 74 69 74 6c	;rt="Ticks";titl
65 3d 22 49 6e 74 65 72 6e 61 6c 20 43 6c 6f 63	e="Internal Cloc
6b 22 3b 63 74 3d 30 2c 3c 2f 61 73 79 6e 63 3e	k";ct=0,</async>
3b 63 74 3d 30	;ct=0

Figure 7: CoAP response message as captured and decoded

The 116 byte unencoded message is shown as ASCII characters in Figure 8 (\xDD stands for the byte with the hex digits DD):

```
bEB\x89\x11(\xA7sometok</>;title="General Info";ct=0,</time>
;if="clock";rt="Ticks";title="Internal Clock";ct=0,</async>;ct=0
```

Figure 8: CoAP response message shown as unencoded characters

The only non-ASCII characters in this example are in the beginning of the message. According to the translation instructions above, the four bytes:

```
89 11 ( A7
```

need the prefixes:

```
2 2 0 1
```

As each prefix character always covers three unencoded bytes, we need the prefix characters for 220 and 100, which are \x1C and \x0B, respectively (Table 2).

The equivalent SMS encoding is shown as equivalent-coded SMS characters in Figure 9 (7 bits per character, \x1C is the 220 prefix and \x0B is the 100 prefix, the rest is shown in equivalent encoding), adding two characters of prefix overhead, for a total length of 118 7-bit characters or 104 (103.25 plus padding) bytes:

```
bEB\x1CI1(\x0B'sometok</>;title="General Info";ct=0,</time>
;if="clock";rt="Ticks";title="Internal Clock";ct=0,</async>;ct=0
```

Figure 9: CoAP response message shown as SMS-encoded characters

Appendix B. Changelog

RFC editor: please remove this appendix.

Changed from draft-05 to draft-06:

- o Update references and addresses
- o Integrate relevant text from coap-misc as an appendix.
- o Section 2 & 3 are merged to section 1

Changed from draft-04 to draft-05:

- o Removed reference to USSD.
- o Updated reference to RFC7252 and 3GPP specs.
- o Updated Options.
- o Adapted URI scheme.

Changed from draft-03 to draft-04:

- o Removed USSD and GPRS related parts.
- o Removed section 5: Examples
- o Removed section 14: Proxying Considerations
- o Added more block size considerations.
- o Added more concatenated SMS considerations.
- o Rewrote encoding scheme section; 7 bit encoding only.

Changed from draft-02 to draft-03:

- o Added reference to OMA LightweightM2M Technical Specification in "Motivation" section.
- o Chose CoAP option numbers and updated the option number table to meet draft-ietf-core-coap-13.

Changed from draft-01 to draft-02:

- o Added security considerations: Transport and Object Security.
Section 11

- o Reply-To-* changed to Response-To-*. Section 12
- o Added URI scheme.
- o Added possible CON/NON/ACK interactions.
- o Added possible M2M proxy scenarios.
- o Added reference to bormann-coap-misc for other SMS encoding. Section 4
- o Updated requirements on Uri-Host and Uri-Port for coap+tel://.
- o Chose CoAP option numbers and updated the option number table to meet draft-ietf-core-coap-10. />
- o Added an IANA registration for the URI scheme. Section 12.2

Acknowledgements

This document is partly based on research for the research project 'The Intelligent Container' which is supported by the Federal Ministry of Education and Research, Germany, under reference number 01IA10001.

The authors of this draft would like to thank Bert Greevenbosch, Marcus Goetting, Nils Schulte and Klaus Hartke for the discussions on the topic and the reviews of this document.

Contributors

Appendix A has been contributed by Carsten Bormann.

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

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

Authors' Addresses

Koojana Kuladinithi (editor)
ComNets, Hamburg University of Technology
Am Schwarzenberg-Campus 3
Hamburg 21073
Germany

Phone: +49 40 428 783533
Email: koojana.kuladinithi@tuhh.de

Markus Becker
Tridonic GmbH & Co KG
Faerbergasse 15
Dornbirn 6851
Austria

Phone: +43 5572 395 45637
Email: markus.becker@tridonic.com

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

Email: kepeng.lkp@alibaba-inc.com

Thomas Poetsch
New York University Abu Dhabi
P.O. Box 129188
Abu Dhabi 129188
United Arab Emirates

Phone: +971 2 628 5069
Email: thomas.poetsch@nyu.edu

CORE WG
INTERNET-DRAFT
Intended Status: Informational
Expires: April 26, 2016

Z. Cao
R. Jadhav
Huawei

October 27, 2016

CoAP Delegated Observe
draft-ca0-core-delegated-observe-00

Abstract

This document discusses the scenarios for "delegated observe", in which a subscriber needs to register some resources on behalf of some other entities. This document also presents a CoAP protocol extension for the delegated observe operation.

Status of this Memo

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

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

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

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

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

Copyright and License Notice

Copyright (c) 2016 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	Scenarios for Delegated Observe	3
2.1	Multiple Devices	3
2.2	Delegation to Cloud	3
2.3	Multicast	4
3	Overview of Delegated Observe	4
4	The Delegated Observe Option	5
5	Handling Delegated Observe	6
6	Security Considerations	6
7	IANA Considerations	6
7	References	7
	Appendix A. Examples	7
	Authors' Addresses	7

1 Introduction

CoAP [RFC7252] is a light-weight application protocol for constrained networks. To avoid keeping polling devices, CoAP supports the 'observe' operation defined in [RFC7641], in which a subscriber can register its interest to certain resources and then be updated with their representation. However, in the current "observe" protocol, the subscriber can only register interests on behalf of itself, and therefore, updated information of the represented resources could not be notified to any parties other than the one who sends the observe request.

This document discusses the scenarios for "delegated observation", in which a subscriber needs to register some resources on behalf of other entities or a group of entities including itself. This document also presents a CoAP protocol extension for the delegated observe operation.

2 Scenarios for Delegated Observe

This section describes scenarios that needs delegated observe.

2.1 Multiple Devices

In a typical smart home network setup, a user with multiple devices wants to observe some sensor resource (e.g., thermometer, bulbs). Instead of sending CoAP Observe requests from every single device, one of the them can send an Observe Registration on behalf of that group of devices, so that all of them will be notified at once.

2.2 Delegation to Cloud

A user wants its mobile device to be notified of a certain sensor information both in-home and off-home. When the device moves out of its smart home network coverage, it is normally hidden behind NAT and FW that keep it being reached for such notification messages. In this case, the normal observe-notification scheme may fail. A walk-around for this case is to let the device send a delegated observe request while at home, asking the home sensors send notifications to the device's representative cloud server, so that the device can always fetch the information from it cloud service while off-home.

This scenario is depicted in Fig. 1.

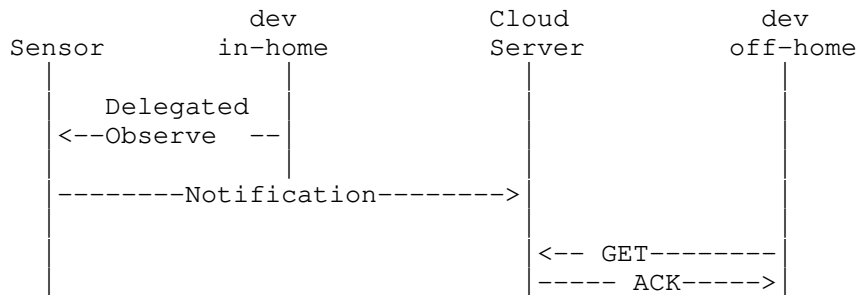


Figure 1: Delegation to Cloud

2.3 Multicast

A group of devices would like to observe the location information on a motion sensor. This is a useful case when a number of light bulbs need to adjust its lighting intensity based on the location of the observed motion object. Instead of let each device register an interest on the motion sensor, one of them could simply delegate the observe to this multicast group, so that the location update notifications will be send to the multicast address that they belong to. This scenario is visualized in Fig.2.

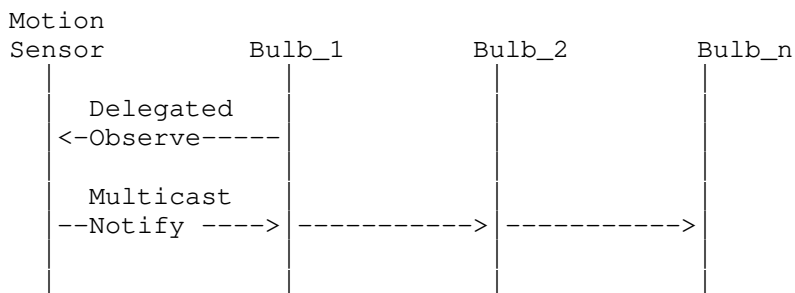


Figure 2: Delegation to a Multicast Group

3 Overview of Delegated Observe

As show in Fig.3, we name the node that has the direct representation of the CoAP resource as the "Source node"; and the immediate node as the 'Delegate node' that will send delegated Observe request to the Source node, on behalf of the "Delegated node" who will be notified by the Source Node.

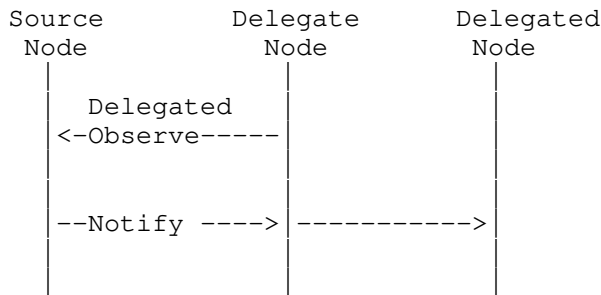


Figure 3: Overview of Delegated Observe

4 The Delegated Observe Option

The properties of the Delegated Observe Option are defined in Fig. 4.

In a GET request:

No.	C	U	N	R	Name	Format	Length	Default
TBD		x	-		Delegated Observe	string	0-256	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

In a Response:

No.	C	U	N	R	Name	Format	Length	Default
TBD		x	-		Delegated Observe	uint	0-3 B	(none)

C=Critical, U=Unsafe, N=No-Cache-Key, R=Repeatable

Figure 4: CoAP Delegated Observe Option

When included in a GET request, the Delegated Observe Option extends the GET method so it does not only retrieve a current representation of the target resource, but also requests the server to add an entry in the list of observers of the resource. This entry maps the target resource to an identifier of the observer contained in the value of this option.

When used to register the representation, the value of Delegated Observe in a GET request is the identifier of the nodes that will be notified, in the format of "IP:port" or "domain_name:port".

When used to de-register the representation, the value of Delegated

Observe in the GET is a special string, e.g., in the format of "::::", the specific format needs further discussion. [TBD]

When included in a response, the Delegated Observe Option identifies the message as a notification. The Option value is used as a sequence number used to infer the order of the notifications. The ordering inference is the same as what has been discussed in Section 3.4 of [RFC7641].

5 Handling Delegated Observe

Before sending the delegated observe, the "Delegate node" needs to know which address:port on the Delegated node will be open to receive the subsequent notification from the source node. This negotiation is out of scope of this document.

Upon receiving the delegated observe request, the "Source Node" will create an entry based on the identifier of the Delegated Node contained within the request. The Source Node can decide if it needs to verify the validity of the Delegated node, per its own policy. The validation way is also out of the scope of this document.

The Delegated Node, once being delegated and verified by Source Node, will be notified subsequently, although it does not send the request for that resource. The Delegated Node interprets the CoAP message, and will handle this message if the CoAP message contains a Delegated Observe option defined in Section 4.

6 Security Considerations

The security considerations in [RFC7252] and [RFC7641] apply.

Delegated observe may increase the risk of amplification attacks, given the source node will send notifications to delegated nodes who have not requested the resource directly. This negative effect can be controlled by several implementation considerations: a) the delegating node can negotiate with the delegated node before sending delegated observe, out of band; b) the source node will strictly control the rate of the notifications, so that flooding will be avoided; c) the delegated node can block any notifications beyond a certain data rate.

7 IANA Considerations

If approved, an Option Number for the defined Delegated Observe Option for CoAP will be needed.

Number	Name	Reference
TBD	Delegated Observe	[thisdoc]

7 References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", RFC 7252, June 2014.
- [RFC7641] K. Hartke, "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, September 2015.

Appendix A. Examples

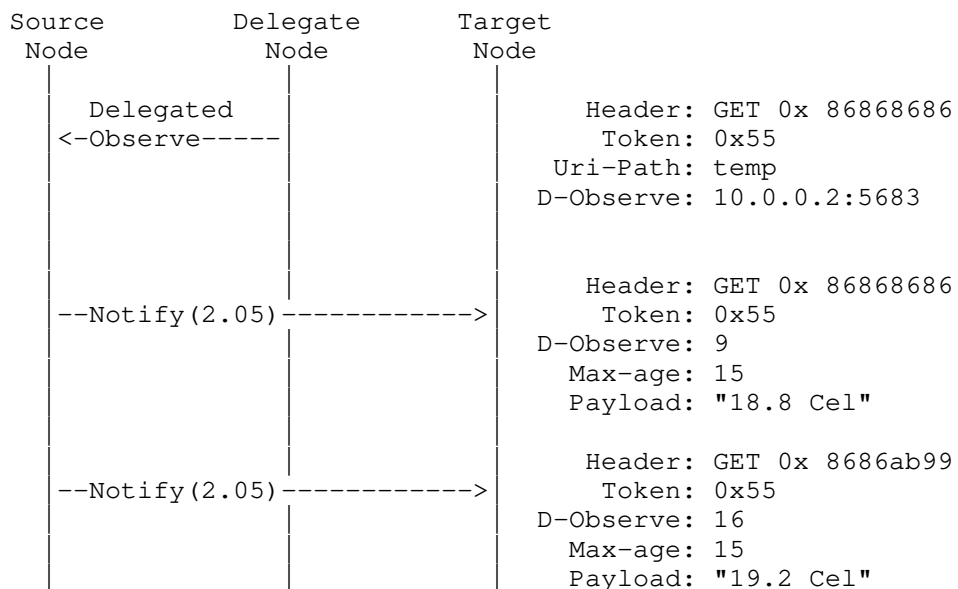


Figure A.1: Example of Delegated Observe

Authors' Addresses

INTERNET DRAFT

draft-cao-core-delegated-observe

<Issue Date>

Zhen Cao
Huawei Tech
Beijing, China

EMail: zhencao.ietf@gmail.com

Rahul Arvind Jadhav
Huawei Tech,
Kundalahalli Village,
Bangalore, India

EMail: rahul.jadhav@huawei.com

CoRE Working Group
Internet-Draft
Intended status: Experimental
Expires: June 9, 2017

D. Garcia
S. Matheu
R. Marin
University of Murcia
December 6, 2016

Application Layer Security for CoAP using the (D)TLS Record Layer
draft-garcia-core-app-layer-sec-with-dtls-record-00

Abstract

This document briefly describes an idea to provide Application-Layer Security for CoAP using (D)TLS Record Layer, assuming it is operative in two CoAP endpoints.

Status of This Memo

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

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

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

This Internet-Draft will expire on June 9, 2017.

Copyright Notice

Copyright (c) 2016 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
1.1. Requirements Language	2
2. Application Layer Security for CoAP with (D)TLS Record Protocol	2
2.1. CoAP Fields to protect	3
3. Processing a CoAP message with the (D)TLS Record	3
4. Bootstrapping the (D)TLS Record Layer for Application Security	6
5. Acknowledgments	7
6. Normative References	7
Authors' Addresses	8

1. Introduction

Secure communications in constrained scenarios is subject of current interest since the restrictions in those kinds of networks motivates rethinking the solutions that up to now have been used in networks that do not suffer from very stringent requirements. (D)TLS [RFC6347][RFC5246] is a standard proposed to secure the communications of CoAP and suitable for end-to-end communications unless a CoAP proxy participates in the communication. To overcome this problem [I-D.ietf-core-object-security] propose Object Security for CoAP (OSCOAP) to allow end-to-end security between two CoAP endpoints in case of a CoAP proxy intermediating between them.

In this document we explore that possibility of providing CoAP security at application layer, assuming a (D)TLS Record Layer is operative (i.e. have the required keys) in both CoAP endpoints. One possibility to "activate" the (D)TLS Record Layer is running (D)TLS handshake over CoAP, as mentioned in CoDTLS [I-D.schmertmann-dice-codtls]. Other (more challenging) options are discussed in [I-D.bhattacharyya-dice-less-on-coap]

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Application Layer Security for CoAP with (D)TLS Record Protocol

To achieve application layer security using the (D)TLS Record, we assume the (D)TLS [RFC6347] [RFC5246] Record layer is already activated, using a protocol such as CoDTLS [I-D.schmertmann-dice-codtls]. Once we have the (D)TLS Record Layer

active, the next step is to define how the CoAP message will be secured end-to-end using the (D)TLS Record Layer.

The entire CoAP message generated by a CoAP sender will need to arrive to the CoAP recipient, achieving integrity and confidentiality for certain parts of the CoAP message (specific CoAP options and payload) excluding the options CoAP intermediaries (proxies) will need to understand to process the CoAP message correctly. The CoAP header would need to arrive maintaining the semantics and version of the protocol.

2.1. CoAP Fields to protect

Here we discuss how the CoAP message is going to be processed to achieve application layer security. How each part of the CoAP message (Header, Options and Payload) is treated and which options are protected and which ones are left unprotected using the (D)TLS Record Layer. Following the procedure specified in OSCOAP [I-D.ietf-core-object-security], we protect all options that are intended to be read by the CoAP recipient.

- o CoAP Header: version and code are protected.
- o CoAP Options: All the options that can be modified by the proxy are left unprotected. All options that are intended for the the CoAP recipient are protected. There might be the case there an option is both left unprotected for the proxy to process and is also intended for the CoAP recipient to see.
- o CoAP Payload: The payload is always protected.

Similarly to OSCON [I-D.ietf-core-object-security], it would be possible to only encrypt the payload of the original CoAP message.

3. Processing a CoAP message with the (D)TLS Record

In this section we analyze how the CoAP message is processed and protected using the (D)TLS Record. In Figure 1 we can see how from the Original CoAP Header we obtain the fields that have to be protected (Version and Code) in 2 bytes. We get the Version and the Code. We will have a padding of 6 bits, then the version and code all in 2 bytes.

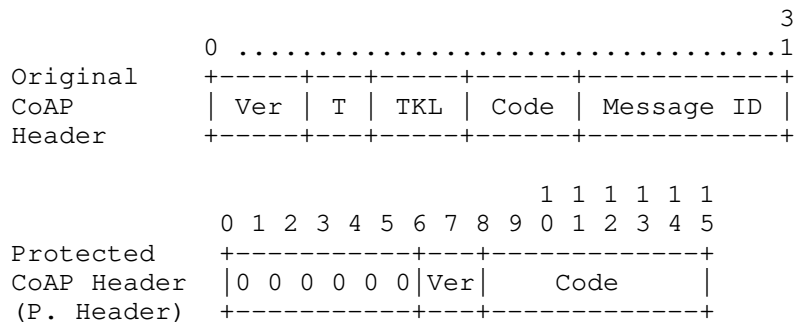
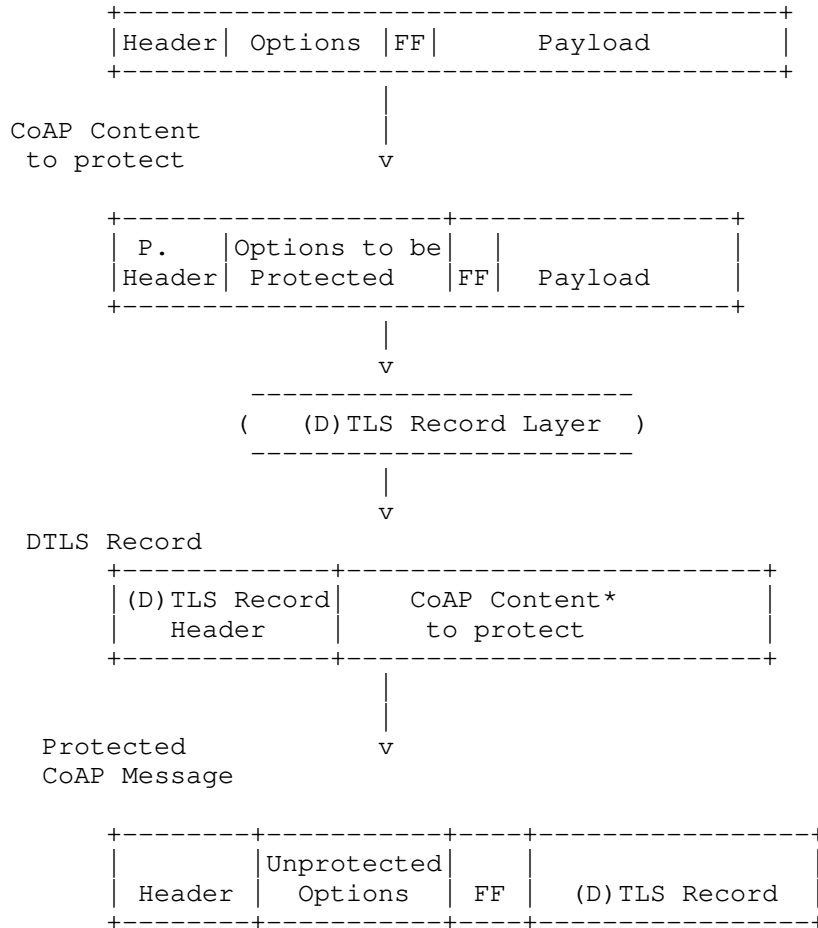


Figure 1: Protected Header Fields

This "denatured" CoAP header is concatenated with the CoAP options to be protected and the Payload (if any) of the Original CoAP message (see Figure 2). This array of bytes is sent to the (D)TLS Record Layer to be processed. The resulting information is a protected array of bytes with a (D)TLS Record Header which will process it generating the (D)TLS Record (adding the (D)TLS Record Header). After that, we add the (D)TLS record to the payload of the message to be sent, that will contain the original CoAP Header and only the options available for the proxies. The Protected CoAP message is formed by the Original CoAP Message Header, the Options that are not considered to be protected with this mechanism, then the marker 0xFF and finally the Payload that contains the (D)TLS Record.

Original CoAP Message



* (Ciphred and Integrity protected)

Figure 2: Processing CoAP message

Upon reception, the CoAP recipient will get the CoAP Payload of the Protected Message and send it to the (D)TLS Record Layer to obtain the Protected Header and the list of options within the (D)TLS Record. With this information, once it is verified correctly, the CoAP recipient constructs the Original CoAP Message.

4. Bootstrapping the (D)TLS Record Layer for Application Security

To enable this solution, the (D)TLS Record Layer in both CoAP endpoints must have a connection to process this information. One alternative is running (D)TLS over CoAP as specified in [I-D.schmertmann-dice-codtls]. However, we consider that it would be possible to define we define a separation between the (D)TLS Handshake and the (D)TLS Record Layer with an interface to be standarized. The (D)TLS Handshake is used to negotiate the parameters to establish a Security Association (SA) in (D)TLS Record Layer. With this interface, we argue that this SA can be set by the (D)TLS Handshake or any other Key Management Protocol (KMP), as we show in Figure 3. This would be similar to the separation in the IPsec architecture [RFC4301], where IKEv2 [RFC7296] is just one of the possible Key Management Protocol to establish IPsec SAs. In fact, IPsec defines a standard API (PFKEY_v2 [RFC2367])) for this purpose.

An example of this separation has been proposed in [I-D.bhattacharyya-dice-less-on-coap]. Another way to benefit from this separation could be that one of the CoAP endpoint has a token (e.g. Kerberos ticket, and ACE token, etc...), with the key material and information (cryptographic keys, algorithms) to start the (D)TLS Record Layer, just presenting the ticket, without the need of running the (D)TLS handshake.

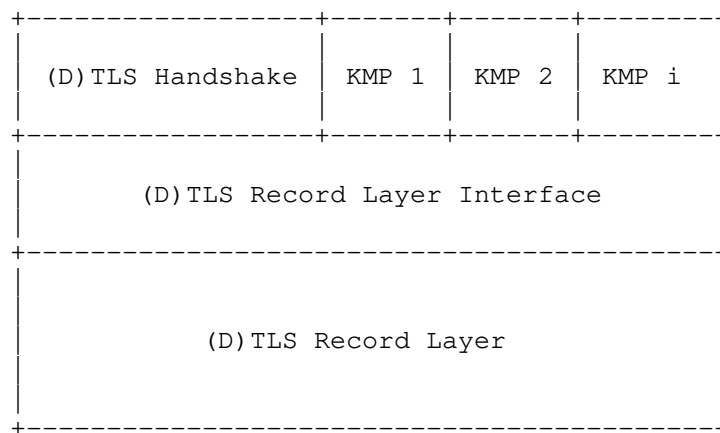


Figure 3: (D)TLS Record Layer Interface

5. Acknowledgments

This work has been possible partially by the ARMOUR project (FP7-ARMOUR-644852 EU Project) and the Spanish National Project CICYT EDISON (TIN2014-52099-R) granted by the Ministry of Economy and Competitiveness of Spain (including ERDF support).

6. Normative References

- [I-D.bhattacharyya-dice-less-on-coap]
Bhattacharyya, A., Bandyopadhyay, S., Ukil, A., Bose, T., and A. Pal, "Lightweight Establishment of Secure Session (LESS) on CoAP", draft-bhattacharyya-dice-less-on-coap-00 (work in progress), April 2015.
- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security of CoAP (OSCOAP)", draft-ietf-core-object-security-00 (work in progress), October 2016.
- [I-D.schmertmann-dice-codtls]
Schmertmann, L., Hartke, K., and C. Bormann, "CoDTLS: DTLS handshakes over CoAP", draft-schmertmann-dice-codtls-01 (work in progress), August 2014.
- [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>>.
- [RFC2367] McDonald, D., Metz, C., and B. Phan, "PF_KEY Key Management API, Version 2", RFC 2367, DOI 10.17487/RFC2367, July 1998, <<http://www.rfc-editor.org/info/rfc2367>>.
- [RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <<http://www.rfc-editor.org/info/rfc4301>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://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, <<http://www.rfc-editor.org/info/rfc6347>>.

- [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>>.
- [RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, DOI 10.17487/RFC7296, October 2014, <<http://www.rfc-editor.org/info/rfc7296>>.

Authors' Addresses

Dan Garcia Carrillo
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia 30100
Spain

Phone: +34 868 88 78 82
Email: dan.garcia@um.es

Sara Nieves Matheu Garcia
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia 30100
Spain

Phone: +34 868 88 78 82
Email: saranieves.matheu@um.es

Rafa Marin-Lopez
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia 30100
Spain

Phone: +34 868 88 85 01
Email: rafa@um.es

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: October 21, 2017

C. Groves
W. Yang
Huawei
April 19, 2017

A WebRTC Data Channel Transport for the Constrained Application Protocol
(CoAP)
draft-groves-coap-webrtc-dc-02

Abstract

The WebRTC framework defines a generic transport service allowing WEB-browsers and other endpoints to exchange generic data from peer to peer utilizing a Stream Control Transmission Protocol (SCTP) transport. This service is known as Web Real Time Communication WebRTC data channels (WebRTC DC). The use of WebRTC DCs for the Constrained Application Protocol (CoAP) allows WebRTC enabled devices to exchange CoAP data between peers in a secure reliable manner.

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 October 21, 2017.

Copyright Notice

Copyright (c) 2017 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. Requirements Language	5
3. Constrained Application Protocol	5
3.1. Message Model	5
3.2. Request Response Model	6
3.3. Intermediaries and Caching	7
3.4. Resource Discovery	7
3.5. Opening Handshake	7
3.6. Message Format	8
3.7. Option Format and Value	9
4. Message Transmission	9
4.1. Messages and Endpoints	10
4.2. Messages Transmitted Reliably	10
4.3. Messages Transmitted without Reliability	11
4.4. Message Correlation	11
4.5. Message Duplication	11
4.6. Message Size	11
4.7. Congestion Control	12
4.8. Transmission Parameters	12
5. Request/Response Semantics	12
6. CoAP URI	12
6.1. coaps+wr URI scheme	13
7. Discovery	13
7.1. Service Discovery	13
7.2. Resource Discovery	14
8. Multicast CoAP	14
9. Securing CoAP	14
10. Interworking	14
11. Security Considerations	15
12. IANA Considerations	15
12.1. New WebRTC DC Protocol Value	15
12.2. Secure Service Name and Port Number Registration	16
12.3. ALPN Protocol ID	16
12.4. URI Schemes	16
12.5. New SIP Media Feature Tag	16
13. Examples	17
14. Acknowledgements	19
15. Changelog	19
16. References	19
16.1. Normative References	19
16.2. Informative References	22

Authors' Addresses	23
------------------------------	----

1. Introduction

Whilst the Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments in constrained network environments its ready adoption has seen the use of it in a multitude of different network environments. For example [I-D.silverajan-core-coap-alternative-transports] provides use cases for alternate CoAP transports.

[I-D.ietf-core-coap-tcp-tls] highlights a number of issues using the native User Datagram Transport (UDP) and envisages deployments more closely integrated with a Web environment. It also proposes the use of the WebSocket protocol [RFC6455]. The use of CoAP over WebRTC DCs has not yet been discussed.

WebRTC is a framework [I-D.ietf-rtcweb-overview] that defines real time protocols for browser-based applications. It allows communications between peer WebRTC endpoints (e.g. browsers) without the need to communicate through a web server.

In addition to protocols for the realtime transport of audio and video, the transport of generic peer-to-peer non-media data has been defined using WebRTC DCs. The non-media data is transported using the Stream Control Transmission Protocol (SCTP) [RFC4960] encapsulated in the Datagram Transport Layer Security (DTLS) [RFC6347]. It allows both reliable and partially reliable transport and provides confidentiality, source authenticated and integrity protected transfers. The use of Interactive Connectivity Establishment (ICE) [RFC5245] allows network address translator (NAT) traversal. The SCTP/DTLS association may be shared with existing audio and video streams enabling multiplexing of several data streams over a single port further facilitating NAT traversal.

Use cases for WebRTC DCs (section 3.1/[I-D.ietf-rtcweb-data-channel]) envisage scenarios where the real-time gaming experience is enhanced by additional object state information. Additional scenarios are considered where information such as heart rate sensor or oxygen saturation sensors could augment audio and video in remote medicine scenarios. The transport of such sensor information is what CoAP has been designed for.

This is illustrated in Figure 1 showing the WebRTC Trapeziod with added sensor/CoAP information. The left hand side WebRTC endpoint acts as a CoAP to CoAP proxy.

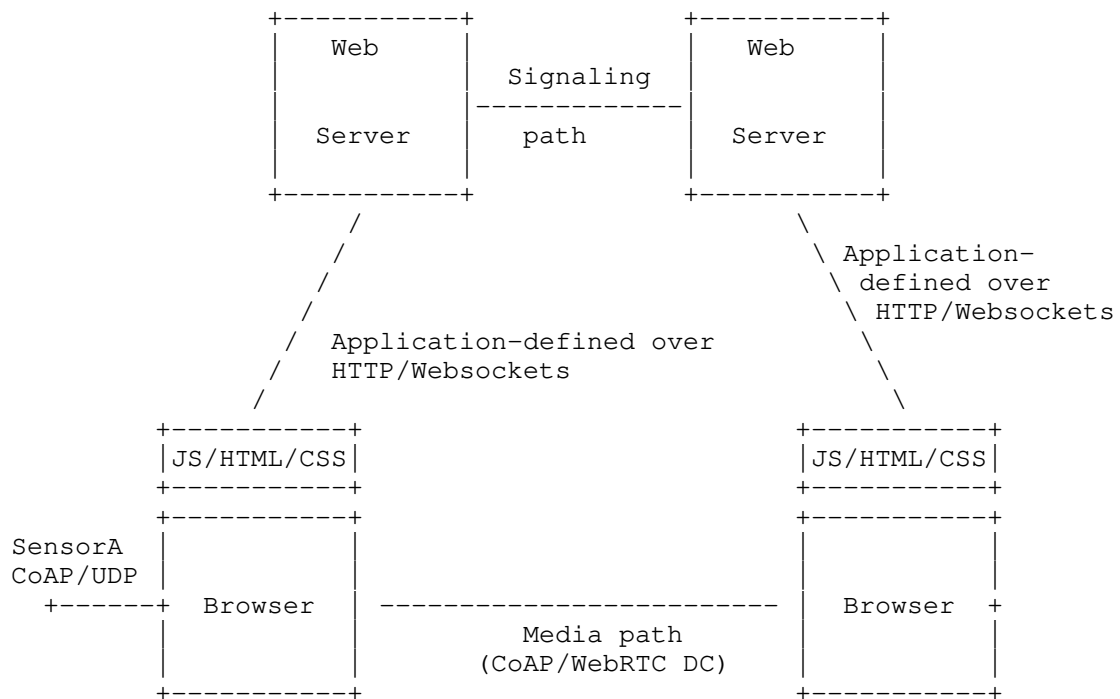


Figure 1: CoAP and WebRTC Trapeziod

By utilizing the WebRTC DC (SCTP over DTLS over ICE/UDP (or ICE/TCP)) transport for CoAP a number of important features are inherited including: congestion control, order and unordered messages delivery, large message transmission by providing segmentation and reassembly and multiple unidirectional streams. A more detailed analysis of the benefits of WebRTC DCs can be found in section 5/[I-D.ietf-rtcweb-data-channel]. [I-D.ietf-tsvwg-sctp-dtls-encaps] describes the usage of SCTP over DTLS.

WebRTC defines in-band and out-of-band methods for establishing a data channel and indicating its characteristics. The Data Channel Establishment Protocol (DCEP) [I-D.ietf-rtcweb-data-protocol] provides an in band means of establishing individual data channels. [I-D.ietf-mmusic-data-channel-sdpneg] uses the Session Description Protocol (SDP) [RFC4566] to provide an out-of-band means to establish data channels.

By defining the use of CoAP over WebRTC DC it negates the need for the WebRTC endpoint to interwork between any CoAP messages received from local devices to a proprietary WebRTC DC format when signalling a remote WebRTC endpoint.

The SCTP Payload Protocol Identifier (PPID) allows the identification of whether a UTF-8 or Binary encoding is being used and thus facilitates the use of text or binary CoAP protocol serializations.

2. Requirements Language

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

3. Constrained Application Protocol

This section describes the use of CoAP over WebRTC DC as a delta to the information contained in section 2/[RFC7252].

Figure 2 shows the CoAP abstract layering as applied to the WebRTC framework.

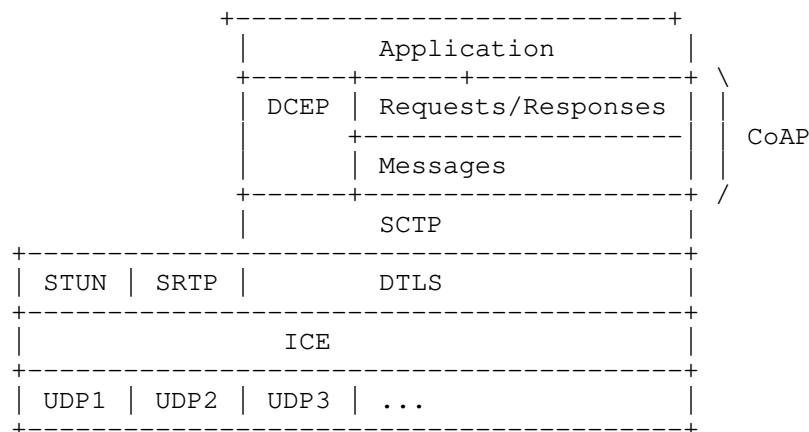


Figure 2: WebRTC protocol layers including CoAP

WebRTC DC mandates the use of SCTP over DTLS. Whilst the above diagram indicates the use of ICE over UDP the use of TCP is also possible in fall back scenarios.

3.1. Message Model

WebRTC DC allows application protocol messages to be exchanged by peers. WebRTC supports both a reliable and partially reliable methods of transmitting user messages.

CoAP [RFC7252] supports four message types "Confirmable, Non-Confirmable, Acknowledge and Reset". As SCTP provides the reliability mechanism the CoAP message types are not needed for CoAP over WebRTC DC.

WebRTC DC does not support multicast usage.

3.2. Request Response Model

WebRTC DCs are realized as a pair of one incoming and one outgoing SCTP stream (with the same identifier) allowing bi-directional communication. Each channel has properties (see section 6.4/[I-D.ietf-rtcweb-data-channel] as discussed below:

- o reliable or unreliable message transmission: WebRTC DCs support the per message indication whether user messages are reliable or partially reliable. Partial reliability indicates that message retransmission is limited to a certain number of retransmissions or lifetime. This loosely parallels to the CoAP usage of Confirmable (CON) or Non-confirmable (NON) messages.
- o in-order or out-of-order message delivery: WebRTC DCs support the per message indication whether user messages are delivered in or out of order. CoAP has been designed for unreliable transports and therefore assumes that messages may arrive out-of-order. CoAP implements a lightweight reliability mechanism to deal with this issue.
- o priority: WebRTC DCs allows a priority to specified for stream scheduling. The usage of this is application specific. Usage of CoAP has no impact on this parameter. It's up to the application using CoAP to set this indication.
- o an optional label: This is an application/implementation specific label. Uniqueness is not guaranteed. Usage of CoAP has no impact on this parameter.
- o an optional protocol: This is used to indicate the application protocol in use. A value is required to identify the usage of CoAP.

As discussed above WebRTC DC supports an unreliable / un-ordered delivery of messages. Implementations utilizing these data channel characteristics may use CoAP messages and request/response model largely unchanged. In this case the CoAP reliability mechanisms would be used. However as WebRTC DC's usage of SCTP is reliable or partially reliable there is some redundancy between the functionality that WebRTC DCs and CoAP provides.

The redundancies are identified and discussed in section 2/[I-D.ietf-core-coap-tcp-tls]. Namely:

1. There is no need to carry acknowledgement semantics at a CoAP level.
2. There is no need for duplicate delivery detection. This is part of the SCTP layer.

3.3. Intermediaries and Caching

As CoAP over WebRTC DC is peer to peer no intermediaries or caching is expected.

3.4. Resource Discovery

The usage of CoAP over WebRTC DC has no foreseeable impacts on resource discovery.

3.5. Opening Handshake

Prior to the establishment of a CoAP over WebRTC DC the characteristics of the SCTP association and data channel may be negotiated by signalling. See Section 4 for further details. For example when using SDP [I-D.ietf-mmusic-sctp-sdp] the use of the "SDP max-message-size" attribute indicates the maximum received SCTP message size.

Further characteristics (such as those described in Section 3.2) are negotiated at the establishment of the WebRTC DC.

On establishment of the CoAP over WebRTC DC the client and server MAY send a CoAP Capability and Settings message (CSM see Section 4.3/[I-D.ietf-core-coap-tcp-tls]) as its first message on the connection to establish CoAP specific capabilities. Any capabilities signalled SHALL not contradict previously negotiated characteristics. Consideration for the individual options are below:

- o Server-Name Setting: CoAP over WebRTC DC clients MAY use the server-name setting option. The initial value is derived based on the signalling method used to establish the WebRTC peer to peer communications. WebRTC does not mandate a signalling method. For example if Websockets is used then the value may be taken from the HTTP host header field.
- o Max-message size Capability: The CoAP Max-Message-Size shall not exceed the SCTP message size.

- o Block-wise Transfer Capability: CoAP over WebRTC DC client and server MAY support the use of BERT (Section 5/[I-D.ietf-core-coap-tcp-tls]). See Section 4.6 for message size considerations.
- o Ping and Pong Messages: Ping and Pong messages MAY be sent by CoAP over WebRTC DC clients and servers. However its use as a basic keepalive is not required as WebRTC defines a method to determine liveness (see Section 4.1).
- o Release Messages: CoAP over WebRTC DC clients and servers may support the CoAP Release message. On receipt of a release message the CoAP over WebRTC DC SHALL be closed as per Section 4.
- o Abort Messages: CoAP over WebRTC DC clients and servers may support the CoAP Abort message. Senders SHALL then close the CoAP over WebRTC DC as per Section 4.

3.6. Message Format

As discussed in [I-D.ietf-core-coap-tcp-tls] the use of a reliable underlying transport allows the use of a modified CoAP header format. The modified format removes the "Type (T)" and "Message ID" fields and introduces a "length" as illustrated below in Figure 3.

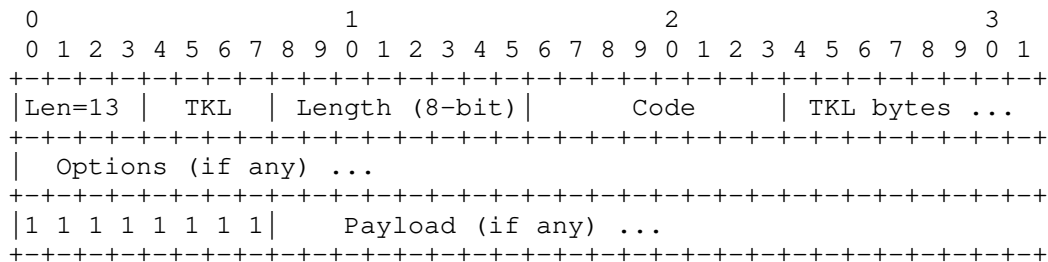


Figure 3: CoAP Header with TCP with 8-bit Length in Header

CoAP over WebRTC DC implementations shall also use the message format in Figure 3 with the following consideration:

- o The length field was added for message delimitation to keep messages separate in TCP. WebRTC DC uses the message orientation of SCTP to preserve message boundaries thus the use of single application message per SCTP user message is mandated by the WebRTC framework. The length field shall be set to 0.

CoAP [RFC7252] supports the use of different content-formats. WebRTC DC defines the use of PPIDs per SCTP user message as follows:

- o WebRTC String: to identify a non-empty JavaScript string encoded in UTF-8.
- o WebRTC Binary: to identify a non-empty JavaScript binary data (ArrayBuffer, ArrayBufferView or Blob).

Depending on the content-format (see section 12.3/[RFC7252]) an appropriate PPID to the encoding type SHOULD be used to minimise the need for translating between encodings. For example content type of "text/plain" would result in the use of PPID "WebRTC String".

Author's note: Specific mappings for each content-format could be provided however given that the formats may change in the future it may be sufficient to offer broad guidance instead.

3.7. Option Format and Value

There are no impacts to option formats or values due to the use of CoAP over WebRTC DCs.

Author's note: Given that the host is determined by the usage of WebRTC are the Uri-Host and Uri-Port relevant? It would seem that this may be valuable to establish a resource tree independent of WebRTC.

4. Message Transmission

In order to use a WebRTC DC, a SCTP over DTLS over ICE/UDP (or ICE/TCP) association must be established. A DTLS connection is established followed by an SCTP association. The out-of-band establishment method through the use of SDP-based Data Channel Negotiation [I-D.ietf-mmusic-data-channel-sdpneg] allows the negotiation of SCTP over DTLS over ICE/UDP as well as the negotiation and establishment of the characteristics of an individual WebRTC DC.

The in-band establishment method through the use of the Data Channel Establishment Protocol (DCEP) [I-D.ietf-rtcweb-data-protocol] only allows for the establishment of a WebRTC DC once the SCTP over DTLS is established. It relies on DATA_CHANNEL_OPEN and DATA_CHANNEL_ACK messages on the relevant SCTP stream to negotiate the properties of the channel. A separate SCTP PPID (50) indicates that the SCTP user message is a WebRTC DCEP message to allow de-multiplexing by the endpoint.

WebRTC DCs are realized as a pair of one incoming and one outgoing SCTP stream (with the same identifier). Requests are sent on an outgoing SCTP stream and received on the peer incoming stream. The SCTP stream identifier is bound to the WebRTC DC instance at the

establishment of the data channel. The establishment protocol provides rules for determining the SCTP stream IDs.

WebRTC DC closure (Stream Reset) is supported through the use of the SCTP stream reconfiguration extension defined in [RFC6525]. The SCTP Stream Reconfiguration reset has the effect of setting the numbering sequence of the SCTP stream back to zero. This is separate function to the CoAP "Reset" message. There is no mapping between the SCTP Stream Reset and the CoAP "Reset" message.

4.1. Messages and Endpoints

As per section 2.5/[I-D.ietf-core-coap-tcp-tls] requests can be sent from both the connecting host and the endpoint that accepted the connection. Who initiated the SCTP/DTLS connection has no bearing on the meaning of the CoAP terms client and server.

WebRTC DC mandates the use of DTLS thus the endpoint is identified depending on the security mode.

WebRTC DCs allows the indication of whether a SCTP user message is empty through the use of PPIDs (WebRTC String Empty and WebRTC Binary Empty). CoAP defines the use of empty messages. However from the perspective of SCTP these CoAP messages would still contain header information thus PPIDs for empty data MUST not be used.

CoAP uses an Empty Confirmable message to provoke a Reset message to check the liveness of an endpoint (so called "CoAP" ping). In WebRTC liveness and the ability to send data is determined through the usage of Session Traversal Utilities for NAT (STUN) Usage for Consent Freshness [RFC7675]. Therefore endpoints utilising CoAP over WebRTC DC MUST not use CoAP "reset" messages.

CoAP also uses Empty messages to acknowledge a request. This is not required due to the SCTP level acknowledgement. Therefore Empty messages MUST not be used with CoAP over WebRTC.

4.2. Messages Transmitted Reliably

For CoAP messages marked as confirmable the sender SHALL use a reliable SCTP user message.

A CoAP endpoint MUST use the ordered delivery SCTP service, as described in [RFC4960], for the CoAP protocol.

CoAP receivers MUST NOT generate CoAP "ACK" or "reset" messages. SCTP level acknowledgement mechanisms are used.

4.3. Messages Transmitted without Reliability

WebRTC DC makes use of the SCTP Partial Reliability (SCTP-PR) Extension [RFC3758]. This extension allows a user to indicate on a per message basis how persistent the transport service should be in attempting to send the message to the receiver. One of the benefits of using this extension identified by [RFC3758] is:

1. Some application layer protocols may benefit from being able to use a single SCTP association to carry both reliable content, - such as text pages, billing and accounting information, setup signaling - and unreliable content, e.g., state that is highly sensitive to timeliness, where generating a new packet is more advantageous than transmitting an old one.

This benefit is also one of the reasons the CoAP "Non-Confirmable" message was introduced. However the SCTP-PR and the CoAP "Non-Confirmable" message mechanisms differs in their approach. The SCTP-PR mechanism focuses on sender side behaviour (e.g. when to abandon retransmission). The CoAP "Non-Confirmable" message focuses on receiver side behaviour (e.g. must not send a CoAP ACK). Even with the use of SCTP-PR an SCTP receiver will send an SCTP level ACK for a successfully received SCTP CHUNK. The CoAP "Non-Confirmable" message has no effect on the SCTP level function.

Therefore the use of a CoAP "Non-Confirmable" message type is redundant as the CoAP receiver will never send a CoAP ACK message in response.

SCTP-PR provides a complimentary function and thus CoAP senders who send Non-confirmable messages SHALL also use SCTP-PR for that message.

4.4. Message Correlation

Due to reliability being handled at the SCTP layers the CoAP "Message ID" is not required.

4.5. Message Duplication

The SCTP layer provides message duplication protection. The CoAP application level procedure is not required.

4.6. Message Size

The considerations in section 4.1/[I-D.ietf-core-coap-tcp-tls] regarding message size limitations also apply to the use of WebRTC DCs. However [I-D.ietf-rtcweb-data-channel] indicates that senders

SHOULD limit the maximum message size to 16KB to avoid monopolization of the SCTP association. Section 5/[I-D.ietf-tsvwg-sctp-dtls-encaps] provides further details regarding segmentation and reassembly and path maximum transmission unit (MTU) discovery.

Interleaving of large user messages is supported by an SCTP protocol extension defined in [I-D.ietf-tsvwg-sctp-ndata].

4.7. Congestion Control

SCTP provides congestion control on a per-association basis (see section 5/[I-D.ietf-rtcweb-data-channel]).

4.8. Transmission Parameters

The application level parameters defined in section 4.8/[RFC7252] are not relevant to SCTP.

5. Request/Response Semantics

Request and response semantics for CoAP over WebRTC DC is as per section 5/[RFC7252] with the following exceptions:

- o section 5.2/[RFC7252]: separate responses MUST be used. Given that WebRTC DC provides an SCTP level acknowledgement it is not possible to piggy back CoAP responses.
- o section 5.3.1/[RFC7252]: due to the use of DTLS the advice regarding token use without using TLS is invalid.
- o section 5.3.2/[RFC7252]: In addition CoAP request/response matching is unique to a particular WebRTC DC (SCTP StreamID pair).
- o section 5.8/[RFC7252]: It is not possible to use a 4.05 piggybacked response.

6. CoAP URI

CoAP [RFC7252] defines the "coap" and "coaps" URI schemes for identifying CoAP resources and providing a means of locating the resource. [RFC7252] defines these resources for use with CoAP over UDP.

Section 8/[RFC7252] (Multicast CoAP), does not apply to the URI schemes defined in the present specification.

Resources made available via the "coaps+wr" schemes have no shared identity with the other scheme or with the "coap" or "coaps" scheme,

even if their resource identifiers indicate the same authority (the same host listening to the same port). The schemes constitute distinct namespaces and, in combination with the authority, are considered to be distinct origin servers.

6.1. coaps+wr URI scheme

```
coaps-wr-URI = "coaps+wr:" "/" host [ ":" port ] path-abempty
               [ "?" query ]
```

The semantics defined in section 6.3/[RFC7252], apply to this URI scheme, with the following changes:

- o The port SHALL be omitted. The underlying UDP or TCP port and SCTP port is negotiated prior to the establishment of the CoAP over WebRTC DC.

7. Discovery

7.1. Service Discovery

WebRTC does not define peer discovery mechanisms. Peers discover each other through the use of the ICE protocol. ICE candidates need to be sent from peer to peer via signalling. The Javascript Session Establishment Protocol (JSEP) [I-D.ietf-rtcweb-jsep] details the generic SDP media descriptions for peer endpoints to determine the characteristics of a session. The actual signalling protocol between application servers is unspecified. WebRTC endpoints MUST implement the network functions detailed by JSEP including ICE functionality.

Whilst the inter-application server signalling protocol is unspecified, the Session Initiation Protocol (SIP) is able to carry SDP for the purposes of establishing a CoAP over WebRTC DC session. SIP allows the use of media feature tags to indicate user agent capabilities [RFC3840]. In order to indicate that a SIP user agent supports the use of CoAP a new "sip.coap" media feature tag is proposed. A CoAP-capable endpoint SHOULD include this media feature tag in its REGISTER requests and OPTION responses. It SHOULD also include the media feature tag in INVITE and UPDATE [RFC3311] requests and responses. Presence of the media feature tag in the contact field of a requestor response can be used to determine that the far end supports CLUE.

The exchange of SDP results in: the underlying transport address (e.g. IPv4 or IPv6), the underlying transport port (e.g. UDP port) the SCTP port and the SCTP StreamID used for the CoAP WebRTC DC being exchanged between the peer endpoints.

7.2. Resource Discovery

On establishment of a CoAP WebRTC DC endpoints are able to use the resource discovery mechanism defined in [RFC6690] for CoAP resources.

8. Multicast CoAP

WebRTC DCs do not support multicast.

9. Securing CoAP

This document defines how to convey CoAP over WebRTC DCs. The WebRTC security architecture [I-D.ietf-rtcweb-security-arch] mandates the use of DTLS for data channels. The use of DTLS 1.2 is compatible with CoAP [RFC7252] which allows makes use of DTLS 1.2.

The use of DTLS for WebRTC is detailed in [I-D.ietf-rtcweb-security-arch].

10. Interworking

An WebRTC endpoint supporting CoAP may in affect act as a gateway between local sensor devices and a remote peer endpoint. The local sensors may utilise CoAP over an alternate signalling transport such as UDP to the local WebRTC endpoint. The WebRTC endpoint may then utilise CoAP over WebRTC to signal to the remote peer.

A CoAP gateway when converting to and from a WebRTC transport will in general perform the following functions:

- o Map received Empty CoAP message to SCTP level operations and discard the empty message.
- o Map received ACK message to SCTP level operations and discard the ACK message.
- o Separate piggy-backed messages.
- o Provide a mapping between received and sent Tokens in order to match requests and responses.

Other behaviour depends on the type of proxy behaviour the gateway is performing. See section 5.7/[RFC7252] for more details.

11. Security Considerations

Security considerations for WebRTC are discussed in [I-D.ietf-rtcweb-security].

The use of CoAP over WebRTC can potentially negate the risks mentioned in:

- o section 11.3/[RFC7252] on insecure UDP and multicast being used to aid an amplification attack.
- o section 11.4/[RFC7252] on IP address spoofing and section 11.5/[RFC7252] on Cross-Protocol attacks.
- o section 11.6/[RFC7252] may also not be relevant as WebRTC endpoints are not expected to be severely constrained.

Of particular relevance to the support of CoAP over WebRTC DC is access to local devices. Devices generating CoAP data are essentially the same as cameras and microphones in that they may expose sensitive data about the user or the location of the device. Thus the guidance of section 4.1/[I-D.ietf-rtcweb-security] applies to devices generating CoAP data. Whilst CoAP has been designed for constrained devices where there is no user interface to inform/request consent, it is assumed that device utilising WebRTC DC for CoAP is more likely at minimum a Class 2 [RFC7228] device that could facilitate consent.

The CoAP media feature tag defined by this document tag may be present in sessions not utilising CoAP, which increases the metadata available about the sending device, which can help an attacker differentiate between multiple devices and help them identify otherwise anonymised users via the fingerprint of features their device supports. To prevent this, SIP signalling SHOULD always be encrypted using TLS [RFC5630].

12. IANA Considerations

12.1. New WebRTC DC Protocol Value

NOTE: This registration is exactly the same as the registration in [I-D.savolainen-core-coap-websockets].

This document requests the registration of the subprotocol name "coap.v1" in the WebSocket Subprotocol Name Registry.

- o Subprotocol Identifier: coap.v1

- o Subprotocol Common Name: Constrained Application Protocol (CoAP)
- o Subprotocol Definition: This document

12.2. Secure Service Name and Port Number Registration

No need has been identified to register a new service name and port number for CoAP over WebRTC. Port number allocation is dynamic. The use of the SCTP over DTLS over UDP/TCP results in a layering of services.

12.3. ALPN Protocol ID

[I-D.ietf-core-coap-tcp-tls] defines a new "coap" application protocol negotiation protocol identity. However as the DTLS connection is used to establish a WebRTC application the protocol identifiers defined in [I-D.ietf-rtcweb-alpn] MUST be used. Note: that confidentiality protection does not extend to WebRTC DCs.

12.4. URI Schemes

This document registers a new URI scheme "coaps+wr" for the use of CoAP over WebRTC DCs. The "coaps+wr" URI schemes can be compared to the "https" URI scheme.

The IANA is requested to add this new URI schemes to the registry established with [RFC7595].

12.5. New SIP Media Feature Tag

This specification registers a new media feature tag in the SIP [RFC3264] tree per the procedures defined in [RFC2506] and [RFC3840].

Media feature tag name: sip.coap

ASN.1 Identifier: 1.3.6.1.8.4.30

Summary of the media feature indicated by this tag: This feature tag indicates that the device supports the Constrained Application Protocol (CoAP).

Values appropriate for use with this feature tag: Boolean.

The feature tag is intended primarily for use in the following applications, protocols, services, or negotiation mechanisms: This feature tag is useful to indicate the support of CoAP.

Related standards or documents: This document

Security Considerations: Security considerations for this media feature tag are discussed in Section 11.

Name(s) and email address(es) of person(s) to contact for further information:

- o CORE workgroup: core@ietf.org
- o CORE chairs: core-chairs@ietf.org

Intended usage: COMMON

13. Examples

The example SDP Offer shows a CoAP over WebRTC DC utilising out-of-band negotiation [I-D.ietf-mmusic-data-channel-sdpneg]. It is based on the example in section 7.2/[I-D.ietf-rtcweb-jsep]. Modified lines are indicated with ">>>" at the start of the line. These indicators are NOT part of the SDP syntax. Note: some lines have been broken into two lines for formatting reasons.

```

v=0
o=- 4962303333179871723 1 IN IP4 0.0.0.0
s=-
t=0 0
a=group:BUNDLE a1 d1
a=ice-options:trickle
m=audio 9 UDP/TLS/RTP/SAVPF 96 0 8 97 98
c=IN IP4 0.0.0.0
a=rtcp:9 IN IP4 0.0.0.0
a=mid:a1
a=msid:57017fee-b6c1-4162-929c-a25110252400
      e83006c5-a0ff-4e0a-9ed9-d3e6747be7d9
a=sendrecv
a=rtpmap:96 opus/48000/2
a=rtpmap:0 PCMU/8000
a=rtpmap:8 PCMA/8000
a=rtpmap:97 telephone-event/8000
a=rtpmap:98 telephone-event/48000
a=maxptime:120
a=ice-ufrag:ATEnlv9DoTMB9J4r
a=ice-pwd:AtSK0WpNtpUjkY4+86js7ZQl
a=fingerprint:sha-256
      19:E2:1C:3B:4B:9F:81:E6:B8:5C:F4:A5:A8:D8:73:04
      :BB:05:2F:70:9F:04:A9:0E:05:E9:26:33:E8:70:88:A2
a=setup:actpass
a=rtcp-mux
a=rtcp-rsize
a=extmap:1 urn:ietf:params:rtp-hdrext:ssrc-audio-level
a=extmap:2 urn:ietf:params:rtp-hdrext:sdes:mid
a=ssrc:1732846380 cname:FocUG1f0fcg/yvY7

m=application 0 UDP/DTLS/SCTP webrtc-datachannel
c=IN IP4 0.0.0.0
a=bundle-only
a=mid:d1
a=fmtp:webrtc-datachannel max-message-size=65536
a=sctp-port 5000
a=fingerprint:sha-256
      19:E2:1C:3B:4B:9F:81:E6:B8:5C:F4:A5:A8:D8:73:04
      :BB:05:2F:70:9F:04:A9:0E:05:E9:26:33:E8:70:88:A2
a=setup:actpass
>>>a=dcmap:0 subprotocol="coap.v1";"label="coap"

```

Figure 4: Example SDP Offer

14. Acknowledgements

We would like to thank the authors of [I-D.ietf-core-coap-tcp-tls] and [I-D.savolainen-core-coap-websockets] for providing a framework for this document. In addition we would like to thank Carsten Bormann for his feedback on message format.

15. Changelog

draft-groves-coap-webrtc-dc-02:

- o Keep alive update. No changes.

draft-groves-coap-webrtc-dc-01:

- o Updated message format to align with draft-core-coap-tcp-tls-04
- o Updates to align with draft-core-coap-tcp-tls-04 as a result of the merger with websockets. Added section on opening handshake. Added support of CoAP capability messages and BERT.

16. References

16.1. Normative References

[I-D.ietf-mmusic-data-channel-sdpneg]

Drage, K., Makaraju, M., Stoetzer-Bradler, J., Ejzak, R., and J. Marcon, "SDP-based Data Channel Negotiation", draft-ietf-mmusic-data-channel-sdpneg-12 (work in progress), March 2017.

[I-D.ietf-mmusic-sctp-sdp]

Holmberg, C., Shpount, R., Loreto, S., and G. Camarillo, "Session Description Protocol (SDP) Offer/Answer Procedures For Stream Control Transmission Protocol (SCTP) over Datagram Transport Layer Security (DTLS) Transport.", draft-ietf-mmusic-sctp-sdp-25 (work in progress), March 2017.

[I-D.ietf-rtcweb-alpn]

Thomson, M., "Application Layer Protocol Negotiation for Web Real-Time Communications (WebRTC)", draft-ietf-rtcweb-alpn-04 (work in progress), May 2016.

[I-D.ietf-rtcweb-data-channel]

Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channels", draft-ietf-rtcweb-data-channel-13 (work in progress), January 2015.

- [I-D.ietf-rtcweb-data-protocol]
Jesup, R., Loreto, S., and M. Tuexen, "WebRTC Data Channel Establishment Protocol", draft-ietf-rtcweb-data-protocol-09 (work in progress), January 2015.
- [I-D.ietf-rtcweb-jsep]
Uberti, J., Jennings, C., and E. Rescorla, "Javascript Session Establishment Protocol", draft-ietf-rtcweb-jsep-20 (work in progress), March 2017.
- [I-D.ietf-rtcweb-overview]
Alvestrand, H., "Overview: Real Time Protocols for Browser-based Applications", draft-ietf-rtcweb-overview-18 (work in progress), March 2017.
- [I-D.ietf-rtcweb-security]
Rescorla, E., "Security Considerations for WebRTC", draft-ietf-rtcweb-security-08 (work in progress), February 2015.
- [I-D.ietf-rtcweb-security-arch]
Rescorla, E., "WebRTC Security Architecture", draft-ietf-rtcweb-security-arch-12 (work in progress), June 2016.
- [I-D.ietf-tsvwg-sctp-dtls-encaps]
Tuexen, M., Stewart, R., Jesup, R., and S. Loreto, "DTLS Encapsulation of SCTP Packets", draft-ietf-tsvwg-sctp-dtls-encaps-09 (work in progress), January 2015.
- [I-D.ietf-tsvwg-sctp-ndata]
Stewart, R., Tuexen, M., Loreto, S., and R. Seggelmann, "Stream Schedulers and User Message Interleaving for the Stream Control Transmission Protocol", draft-ietf-tsvwg-sctp-ndata-09 (work in progress), March 2017.
- [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>>.
- [RFC2506] Holtman, K., Mutz, A., and T. Hardie, "Media Feature Tag Registration Procedure", BCP 31, RFC 2506, DOI 10.17487/RFC2506, March 1999, <<http://www.rfc-editor.org/info/rfc2506>>.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, DOI 10.17487/RFC3264, June 2002, <<http://www.rfc-editor.org/info/rfc3264>>.

- [RFC3311] Rosenberg, J., "The Session Initiation Protocol (SIP) UPDATE Method", RFC 3311, DOI 10.17487/RFC3311, October 2002, <<http://www.rfc-editor.org/info/rfc3311>>.
- [RFC3758] Stewart, R., Ramalho, M., Xie, Q., Tuexen, M., and P. Conrad, "Stream Control Transmission Protocol (SCTP) Partial Reliability Extension", RFC 3758, DOI 10.17487/RFC3758, May 2004, <<http://www.rfc-editor.org/info/rfc3758>>.
- [RFC3840] Rosenberg, J., Schulzrinne, H., and P. Kyzivat, "Indicating User Agent Capabilities in the Session Initiation Protocol (SIP)", RFC 3840, DOI 10.17487/RFC3840, August 2004, <<http://www.rfc-editor.org/info/rfc3840>>.
- [RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, DOI 10.17487/RFC4566, July 2006, <<http://www.rfc-editor.org/info/rfc4566>>.
- [RFC4960] Stewart, R., Ed., "Stream Control Transmission Protocol", RFC 4960, DOI 10.17487/RFC4960, September 2007, <<http://www.rfc-editor.org/info/rfc4960>>.
- [RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, DOI 10.17487/RFC5245, April 2010, <<http://www.rfc-editor.org/info/rfc5245>>.
- [RFC5630] Audet, F., "The Use of the SIPS URI Scheme in the Session Initiation Protocol (SIP)", RFC 5630, DOI 10.17487/RFC5630, October 2009, <<http://www.rfc-editor.org/info/rfc5630>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.
- [RFC6525] Stewart, R., Tuexen, M., and P. Lei, "Stream Control Transmission Protocol (SCTP) Stream Reconfiguration", RFC 6525, DOI 10.17487/RFC6525, February 2012, <<http://www.rfc-editor.org/info/rfc6525>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://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, <<http://www.rfc-editor.org/info/rfc7228>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<http://www.rfc-editor.org/info/rfc7252>>.
- [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, <<http://www.rfc-editor.org/info/rfc7595>>.
- [RFC7675] Perumal, M., Wing, D., Ravindranath, R., Reddy, T., and M. Thomson, "Session Traversal Utilities for NAT (STUN) Usage for Consent Freshness", RFC 7675, DOI 10.17487/RFC7675, October 2015, <<http://www.rfc-editor.org/info/rfc7675>>.

16.2. Informative References

- [I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-07 (work in progress), March 2017.
- [I-D.savolainen-core-coap-websockets]
Savolainen, T., Hartke, K., and B. Silverajan, "CoAP over WebSockets", draft-savolainen-core-coap-websockets-07 (work in progress), June 2016.
- [I-D.silverajan-core-coap-alternative-transports]
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-09 (work in progress), December 2015.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<http://www.rfc-editor.org/info/rfc6455>>.

Authors' Addresses

Christian Groves

Email: cngroves.std@gmail.com

Weiwei Yang

Huawei

Email: tommy@huawei.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 14, 2017

C. Groves
W. Yang
Huawei
March 13, 2017

Binding Attribute Scope
draft-groves-core-bas-01

Abstract

This document specifies an additional CoAP binding attribute that allows binding attributes to be scoped to an item (sub-resource) in a collection resource. This allows synchronisation of multiple resources in a collection based on the value of one resource.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 14, 2017.

Copyright Notice

Copyright (c) 2017 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. Requirements Language	2
2. Introduction	2
2.1. Usage of a collection	3
2.2. Multiple Conditions	3
3. Terminology	4
4. Binding Attributes	4
4.1. Binding Attribute Scope	5
4.2. Interactions	5
4.3. Examples	5
4.3.1. Example 1 - Item Binding Attribute	6
4.3.2. Example 2 - Multiple Observes	6
5. Security Considerations	6
6. IANA Considerations	6
7. Acknowledgements	6
8. Changelog	7
9. Normative References	7
Authors' Addresses	8

1. Requirements Language

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

2. Introduction

In some cases a CoAP client may require a notification of the value of a resource based on some condition associated with another resource. For example: a client wants to be notified of the machine state once a temperature sensor threshold is crossed. Currently the only way to implement this is for a client to be notified of the temperature resource change and then for the client to do a separate read of the required resources. It would be more efficient to provide the information in a single response.

This document defines a new "Binding Attribute Scope" binding attribute to allow a client to request collection resource state synchronisation based on a conditional value of an item in the collection resource.

The "Binding Attribute Scope" (bas) attribute is set on a collection and is used to indicate which item in a collection an Observe and associated binding attributes apply to. A linked batch (or batch) interface (sect.4.3/[I-D.ietf-core-interfaces]) is used on the collection. The client constructs a collection resource using the

linked batch interface or uses a pre-provisioned collection via the batch interface to set the information that it requires. By placing the required information in a collection it allows a GET on the collection to return all the information based on the conditional value of an item.

Section 4 below indicates the text that would need to be added to [I-D.ietf-core-dynlink] to add the Binding Attribute Scope attribute.

2.1. Usage of a collection

[I-D.ietf-core-interfaces] indicates that a collection may be observed. An observation returns a representation of the entire collection. The "bas" binding attribute is only used on collection (e.g. batch, linked-batch) resources which enables the use of the binding attributes from [I-D.ietf-core-dynlink] on a collection. The "bas" attribute points to one item in the collection.

This approach does have the downside that it requires the use of a collection interface even if a single resource is required. The scope of the synchronization is limited to the current collection. However the use of a collection to return the information does seem to fit with the principle that a GET of a resource should return the same representation as a notification.

2.2. Multiple Conditions

Given that the "bas" binding attribute only applies to one item/sub-resource in a collection it means that multiple resources cannot be used as conditions for a binding synchronization. To allow the use of conditions on multiple resources to determine resource synchronization a different approach would be required.

One approach would be to use the FETCH method [I-D.ietf-core-etch] with a set of request parameters. A content type could be defined that allowed the binding attributes to be specified on multiple resources.

For example:

```
FETCH /s/?pmin=1&pmax=100 content-type=application/conditionals+json
```

```
[
  {
    "n": "/s/light",
    "st": 5
  },
  {
    "n": "/s/temp",
    "st": 1
  },
  {
    "n": "/s/humidity",
    "lt": 40,
    "gt": 70
  }
]
```

This approach would add complexity over the use of the bas binding attribute however such complexity may be warranted in certain use cases. A new content-format would need to be defined. This approach is for further study.

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "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].

4. Binding Attributes

This document introduces one new attributes:

Attribute Name	Parameter	Data Format
Binding Attribute Scope	bas	xsd:string

Table 1: Binding Attribute Summary

The xsd:string contains a Uri-Path.

4.1. Binding Attribute Scope

The binding attribute scope attribute allows a client to indicate that the binding attributes contained in the query apply to a certain item in a collection resource.

The collection containing the Observed resource and the additional resources to be returned is first created through the use of the Batch or Linked-Batch [I-D.ietf-core-interfaces] interface. The Linked-Batch allows a client to dynamically associate resources.

The client then creates a binding (e.g. an Observe) with the collection indicating the required binding attributes (e.g. "gt", "lt" etc.) and to which sub-resource these apply through the use of the "bas" binding attribute containing a URI-path.

A client may create multiple bindings for the collection. When using Observe a client may use multiple GET Observes with different query parameters. Each observation is identified through the use of different Tokens.

If the server determines that the "bas" attribute contains an unknown URI-path then it responds with response code 4.04 "Not Found".

State synchronisation occurs when the sub-resource (item) value meets the conditions indicated by the binding attributes. When state synchronisation occurs the collection resource (including sub-resources) will be synchronised.

4.2. Interactions

The "bas" parameter modifies the scope of other binding attributes in a query to apply to the resource specified in the "bas" URI-path.

Editor's note: the interaction with sample time window/sample number window needs to be added if accepted.

4.3. Examples

Given the resource links:

```
Req: GET /.well-known/core
Res: 2.05 Content (application/link-format)
</s/>;rt="simple.sen";if="core.b",
</s/light>;rt="simple.sen.light";if="core.s",
</s/temp>;rt="simple.sen.tmp";if="core.s";obs,
</s/humidity>;rt="simple.sen.hum";if="core.s"
```

4.3.1. Example 1 - Item Binding Attribute

A Req: GET /s?bas="temp">37
Token: 0x4a
Observe: 0
would produce the following when temp exceeds 37:

```
Res: 2.05 Content (application/senml+json)
Token: 0x4a
{"e":[
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 38, "u": "degC" },
  { "n": "/s/humidity", "v": 80, "u": "%RH" }],
}
```

4.3.2. Example 2 - Multiple Observes

In addition to the GET in example 1 the client could also request a notification when the humidity raises above 90%.

A Req: GET /s?bas="humidity">90
Token: 0x4b
Observe: 0
would produce the following when temp exceeds 37:

```
Res: 2.05 Content (application/senml+json)
Token: 0x4b
{"e":[
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 16, "u": "degC" },
  { "n": "/s/humidity", "v": 92, "u": "%RH" }],
}
```

Editor's note: Need to add example for pmin

5. Security Considerations

As per 5/[I-D.ietf-core-dynlink].

6. IANA Considerations

None.

7. Acknowledgements

Michael Koster for his comments and suggestion of the FETCH approach.
Friedhelm Rodermund for stimulating discussion of the use case.

Mojan Mohajer for raising the multiple observes on a collection use case.

8. Changelog

draft-groves-core-bas-01

- o Section 2: Removed editor's note on alternate solution
- o Section 2.1: Changed language around observe on collections.
- o Section 4.1: Added text regarding multiple observations.

draft-groves-core-bas-00

- o Initial version

9. Normative References

[I-D.ietf-core-dynlink]

Shelby, Z., Vial, M., Koster, M., and C. Groves, "Dynamic Resource Linking for Constrained RESTful Environments", draft-ietf-core-dynlink-02 (work in progress), February 2017.

[I-D.ietf-core-etch]

Stok, P., Bormann, C., and A. Sehgal, "Patch and Fetch Methods for Constrained Application Protocol (CoAP)", draft-ietf-core-etch-04 (work in progress), November 2016.

[I-D.ietf-core-interfaces]

Shelby, Z., Vial, M., Koster, M., and C. Groves, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-08 (work in progress), February 2017.

[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>>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<http://www.rfc-editor.org/info/rfc5988>>.

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

Authors' Addresses

Christian Groves
Huawei
Australia

Email: cngroves.std@gmail.com

Weiwei Yang
Huawei
P.R.China

Email: tommy@huawei.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 25, 2017

C. Groves
W. Yang
Huawei
February 21, 2017

Additional CoAP Binding and Observe Attributes
draft-groves-core-obsattr-00

Abstract

[I-D.ietf-core-dynlink] defines five CoAP Observaton attributes (minimum period, maximum period, band step, less than and greater than) to control when notifications are sent. These attributes are insufficient for some use cases. This document specifies additional attributes allowing for notification bands, initialization values, band step, sample number window and sample time window to allow for a wider range of use cases.

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 August 25, 2017.

Copyright Notice

Copyright (c) 2017 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. Requirements Language	2
2. Introduction	2
3. Terminology	4
4. Binding and Resource Observation Attributes	4
4.1. Initialization Value (iv)	5
4.2. Notification Band Minimum (bmn)	6
4.3. Notification Band Maximum (bmx)	6
4.4. Band Step (bst)	6
4.5. Sample Number Window	7
4.6. Sample Time Window	7
4.7. Interactions	8
4.8. Examples	9
4.8.1. Example 1 - Band Minimum and Maximum	9
4.8.2. Example 2 - Band Minimum and Maximum and Step	10
4.8.3. Example 3 - Band Minimum and Maximum, Step and Initialization Value	10
4.8.4. Example 4 - Step and Initialization Value	11
4.8.5. Example 5 - Band Minimum and Maximum, Band Step and Initial Value	11
4.8.6. Example 6 - Band Minimum and Sample Number Window	12
5. Security Considerations	12
6. IANA Considerations	12
7. Acknowledgements	12
8. Changelog	12
9. References	12
9.1. Normative References	12
9.2. Informative References	13
Authors' Addresses	13

1. Requirements Language

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

2. Introduction

[I-D.ietf-core-dynlink] defines five CoAP Binding and Observation attributes (minimum period, maximum period, change step, less than and greater than) to control when notifications are sent. The currently defined attributes have characteristics that means some use cases cannot be supported. These are described below:

- o A transition across a less than (lt), or greater than (gt) limit or a change step (st) 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.
- o The change step (st) value is not deterministic when setting the attribute. A client cannot set the initial value that the change step applies to. The change step is based on the initial value sent by the server. This means that a client cannot indicate that it wants the change step (st=10) to apply to a certain increment (e.g. 10, 20, 30) instead it relies on the initial value from the client. Thus a change step of 10 could result in reporting (11, 21, 31) or equally (15, 27, 53).
- o SenML allows for multiple values (records) to be reported for a resource. The current attributes do not allow a method for a client to request a particular number of records or sample time window.

In order to allow a more complete set of usecases to be supported this specification introduces several new attributes.

The notification band attributes "Notification Band Minimum" (bmn) and "Notification Band Maximum" (bmX) attributes allow 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.

An "Initialization Value" (iv) attribute allows a seed value for the calculation of the change step to be specified. This allows use cases where synchronization occurs around a known value. For example: synchronization will occur based on the operating temperature set point of a machine. Without the initialization synchronization will occur around the first measured value.

A "Band Step" (bst) attribute defines a series of bands that will trigger state synchronization. This allows use cases where state synchronization is required against known levels. Rather than synchronizing based on a difference to a previous synchronization value, synchronization occurs against a fixed known level. For example: it allows state synchronisation for a sensor when it's value is between (5,10], (10,15], (15,20] etc.

The "Sample Number Window" (snw) attribute allows a number of state synchronizations to be queued before the actual queue synchronization occurs. Once the number of queued state synchronizations has reached a certain level then a single queue synchronization occurs with the multiple resource values related to individual state synchronizations included. This allows use cases where multiple resource values are required but frequent synchronization is not required as there is a need to minimise resource usage. For example: a meter may need to be recorded once an hour but the values only need to be synchronized once a day.

The "Sample Time Window" (stw) attribute as per the sample number window allows state synchronizations to be queued before the actual queue synchronization occurs. The queue synchronization occurs when the indicated period expires independent of the number of samples.

3. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "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]. This specification makes use of the following additional terminology:

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.

4. Binding and Resource Observation Attributes

The attributes defined in this section are additional attributes that may be used as binding attributes (3.3/[I-D.ietf-core-dynlink]) and resource observation attributes (4.2/[I-D.ietf-core-dynlink]).

This specification introduces several new attributes:

Attribute Name	Parameter	Data Format
Initialialization Value	/ {resource} ?iv	xsd:decimal
Band Minimum Notification	/ {resource} ?bmn	xsd:decimal
Band Maximum Notification	/ {resource} ?bmx	xsd:decimal
Band Step	/ {resource} ?bst	xsd:decimal (>0)
Sample Number Window	/ {resource} ?snw	xsd:integer (>0)
Sample Time Window	/ {resource} ?stw	xsd:integer (>0)

Table 1: Resource Observation Attribute Summary

The attributes may only be included at most once in a query.

4.1. Initialization Value (iv)

The attribute indicates the initialization value to be used to determine when a change step is notified. As such it MUST only be present in a query when the change step (st) or band step (bst) attribute is present (see [I-D.ietf-core-dynlink]). If st or bst is not present then the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

Without iv, on reception of a query the synchronization initiator uses the current value for the observed resource as the initial value to which the change step is applied. The use of iv overrides this behaviour and the iv value is used for the initial value (STinit or BSTinit). A state synchronization occurs once the resource value differs from the initial value by the change step value (i.e. $\text{CurrVal} \geq \text{STinit} + \text{ST}$ or $\text{CurrVal} \leq \text{STint} - \text{ST}$). The initial value is then set to the state synchronization value.

A state synchronization due to Pmax (or Pmin) does not cause an update of the initial value. However once the initial value is updated by a state synchronization due to the other attributes in the query then the normal behaviour defined by 3.3.7/[I-D.ietf-core-dynlink] occurs.

4.2. Notification Band Minimum (bmn)

This attribute defines the lower bound for the notification band. State synchronization occurs when the resource value is equal to or above the notification band minimum. This attribute is optional. If not present there is no minimum value for the band. If present bmn must be less than bmx if it is also present otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

4.3. Notification Band Maximum (bmx)

This attribute defines the upper bound for the notification band. State synchronization occurs when the resource value is equal to or less than the notification band maximum. This attribute is optional. If not present there is no maximum value for the band. If present bmx must be more than bmn if it is also present, otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

4.4. Band Step (bst)

Like change step (st) this attribute indicates how much the value of a resource SHOULD change before triggering a state synchronization. The difference however is that the values used for the band step calculation are based on a constant step rather than being based on the synchronized value.

The current resource value or the initialization value (if provided) is used as a seed to determine the band thresholds.

For example: Given a bst=10 and an initialization value=25. This defines a series of band step thresholds: i.e. ..., (5,15], (15,25], (25,35], ...

When the resource value enters a new band step by exceeding the minimum threshold value and being less than or equal to the maximum threshold value for a band step then synchronisation occurs.

A new synchronization occurs whenever the value enters a new band step. If the value jumps across band steps e.g. from 13 to 27 only one synchronisation occurs.

The band step MUST be greater than zero otherwise the server MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

Change step (st) and band step (bst) MUST NOT occur together in the same query.

4.5. Sample Number Window

If queuing of a number of state synchronizations are required then the sample number window attribute is set to the desired size of the window. The attribute may be set with valid combinations of other binding/resource observation attributes. When a state synchronization is triggered due to the other attributes the resource value is added to the list of samples instead of resulting in an update of the source and destination resource (state synchronization). Only when the number of samples in the window reaches the sample number window is a state synchronisation performed for the resource. The samples are then flushed from the window and the process is repeated.

The use of the sample number window attribute may require the use of a suitable content-format (such as SenML [I-D.ietf-core-senml]) that allows multiple values/data points to be specified during the state synchronization.

Consideration should also be given to the resource capacity (i.e. memory) of the CoAP server for storing data associated with the sample window. The sample number window should not exceed its capabilities. Even if the sample number window has not been reached, if resource (memory) consumption is an issue then state synchronization for the stored resource values SHOULD occur enabling resources to be freed.

The pmin and pmax attributes have an indirect effect on the overall state synchronization. Whilst pmin and pmax do not directly specify the period for the overall state synchronization the setting of pmin and pmax may trigger samples entering the sample window as thus affect the frequency of state synchronization.

4.6. Sample Time Window

If state synchronizations are to be queued during a certain period of time (in seconds) then the sample time window attribute is used. The attribute may be set with valid combinations of other binding/resource observation attributes. On reception of a query with the stw attribute a timer (T1=0) is started. Whilst T1<stw when a state synchronization is triggered due to the other attributes, the resource value is added to the sample window instead of resulting in a state synchronization. When the time expires e.g. T1=stw the state synchronization for the resource occurs. The window is then flushed, T1 is re-started and the process is repeated.

The use of the sample time window attribute may require the use of a suitable content-format (such as SenML [I-D.ietf-core-senml]) that

allows multiple values/data points to be specified during the synchronization.

Consideration should also be given to the resource capacity (i.e. memory) of the CoAP server for storing data associated with the time window. Consideration should be given to that the expected frequency of adding resource values and length of the time doesn't exceed the memory capacity of the server. Even if the sample time window has not been reached, if resource (memory) consumption is an issue then state synchronization for the stored resource values SHOULD occur enabling resources to be freed.

4.7. Interactions

To enable an notification band at least bmn or bmx MUST be set. If both bmn and bmx are set then a finite band is specified. State synchronization occurs whenever the resource value is between bmn and bmx or is equal to bmn or bmx. If only one attribute bmx or bmn is set then the band has an open bound. That is all values above bmn or all values below bmx will be synchronized.

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=20) and BandB (Bbmn=21, Bbm=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.

Note: The use of bmn and bmx 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.

If pmin and pmax are present in a query then they take precedence over the other parameters. Thus even if bmn and bmx are met if pmin has not been exceeded then no state synchronization occurs. Likewise if bmn and bmx have not been met and pmax time has expired then state synchronization occurs. The current value of the resource is used for the synchronization. If pmin time is exceeded and bmn and bmx are met then the current value of the resource is synchronized. If st is also included, a state synchronization resulting from pmin or pmax updates STinit with the synchronized value. If bst is included, a state synchronization resulting from pmin or pmax updates bstinit with the closest bst delta value as per Section 4.4.

It is an error to include a greater than (gt) or less than (lt) attribute in a query containing bmn or bmx.

If change step (st) is included in a query with bmn or bmx then state synchronization will occur whilst the resource value is in the notification band AND the resource value differs from STinit by the change step.

If band step (bst) is included in a query with bmn or bmx then state synchronization will occur whilst the resource value is in the notification band defined by bmn or bmx AND has entered a new bandstep band.

If bst is included in a query with a gt or lt attribute then state synchronizations occur only when the conditions described by bst AND gt or bst AND lt are met.

If bst is included in a query with a iv attribute then iv is used to calculate the band thresholds. Subsequent state synchronizations are as per Section 4.4.

If iv is included with the bmn and bmx or gt and lt attributes it has no affect on the synchronisation. If bst is also used then it used to calculate band step thresholds. If st is instead used then it is used to calculate the initial value.

The snw and stw attributes SHALL not be used in the same query together. Synchronizations based on pmin and pmax are added to the snw/stw sample window. In effect this overrides the pmin and pmax mechanism because resource state synchronizations will not occur between the source and destination resources based on these parameters. For snw the minimum period will be snw * pmin. The maximum period will be snw * pmax. For stw the state synchronization will occur after time stw and is not dependent on pmin and pmax (unless a state synchronization occurs due to memory constraints). The stw must be more than pmax if it is present, otherwise the pmax attribute becomes invalid.

4.8. Examples

4.8.1. Example 1 - Band Minimum and Maximum

```
Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/temperature>; rel="boundto"; anchor="/a/
temperature"; bind="obs"; pmin="10"; pmax="60"; bmn="20", bmx="40"
```

```
Res: 2.04 Changed
```

The above will result in a state synchronization through an Observe:

- o Every 60 seconds if the value is not between 20 and 40.
- o When the temperature is equal to or between 20 and 40 at least every 10 seconds.

4.8.2. Example 2 - Band Minimum and Maximum and Step

```
Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/temperature>; rel="boundto"; anchor="/a/
temperature"; bind="obs"; pmin="10"; pmax="60"; bmn="20", bmx="40", st="5"
```

Res: 2.04 Changed

The above will result in:

- o STinit being set to the temperature value at the time of the POST.
- o A state synchronization through an Observe:
 - * Every 60 seconds if the value is not between 20 and 40 and if the value has not changed by 5
 - * When the temperature is equal to or between 20 and 40 and the value has changed by 5 from STinit at least every 10 seconds.

4.8.3. Example 3 - Band Minimum and Maximum, Step and Initialization Value

```
Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/temperature>; rel="boundto"; anchor="/a/t
emperature"; bind="obs"; pmin="10"; pmax="60"; bmn="20", bmx="40", st="5", i
v="20"
```

Res: 2.04 Changed

The above will result in:

- o STinit being set to 20 due to iv.
- o A state synchronization through an Observe:
 - * Every 60 seconds if the value is not between 20 and 40 and if the value has not changed by 5 from 20
 - * When the temperature is equal to or between 20 and 40 and the value has changed by 5 from STinit at least every 10 seconds.

4.8.4. Example 4 - Step and Initialization Value

```
Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/temperature>; rel="boundto"; anchor="/a/
temperature"; bind="obs"; pmin="10"; pmax="60"; st="5", iv="20"
```

Res: 2.04 Changed

The above will result in:

- o STinit being set to 20 due to iv.
- o A state synchronization through an Observe:
 - * Every 60 seconds if the temperature does not differ from STinit by 5.
 - * When the temperature differs from STinit by 5 at least every 10 seconds.

4.8.5. Example 5 - Band Minimum and Maximum, Band Step and Initial Value

```
Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/temperature>; rel="boundto"; anchor="/a/t
emperature"; bind="obs"; pmin="10"; pmax="60"; bmn="20", bmx="40", bst="5",
iv="15"
```

Res: 2.04 Changed

The above will result in:

- o A series of bands being created of a width of 5 with the seed value 15. Given bmn="20" and bmx="40" this effectively means that bands with the following thresholds (15,20], (20,25], (25,30], (30,35], (35,40] are created.
- o A state synchronization through an Observe:
 - * Every 60 seconds if the value is not between 20 and 40 (inclusive) and if the value has not entered into a new band.
 - * When the temperature is equal to or between 20 and 40 and the value changes between the bands at least every 10 seconds.

4.8.6. Example 6 - Band Minimum and Sample Number Window

```
Req: POST /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/temperature>; rel="boundto"; anchor="/a/
temperature"; bind="obs"; pmin="10"; pmax="60"; bmn="50"; snw="5"
```

Res: 2.04 Changed

The above will result in:

- o A state synchronization added to the queue at pmax or whenever the value changes and is equal to or above 50.
- o A state synchronization through an Observe occurring once 5 synchronizations have been added to the queue resulting in multiple values being synchronized between the source and destination resources.

5. Security Considerations

As per 5/[I-D.ietf-core-dynlink].

6. IANA Considerations

None.

7. Acknowledgements

Michael Koster for discussions leading to the creation of these notification band and initialization attributes.

8. Changelog

draft-groves-core-intparam-00

- o Initial version

9. References

9.1. Normative References

[I-D.ietf-core-dynlink]
Shelby, Z., Vial, M., Koster, M., and C. Groves, "Dynamic Resource Linking for Constrained RESTful Environments", draft-ietf-core-dynlink-01 (work in progress), October 2016.

- [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>>.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<http://www.rfc-editor.org/info/rfc5988>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.

9.2. Informative References

- [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-04 (work in progress), October 2016.

Authors' Addresses

Christian Groves
Huawei
Australia

Email: Christian.Groves@mail01.huawei.com

Weiwei Yang
Huawei
P.R.China

Email: tommy@huawei.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 21, 2017

C. Groves
W. Yang
Huawei
April 19, 2017

Addition of organisation prefix to RFC6690 IANA CoRE parameters
registration
draft-groves-core-rfc6690up-01

Abstract

[RFC6690] defines the resource type 'rt' and interface description 'if' link attributes and defines procedures for registering values. Currently each 'rt' and 'if' attribute value must be registered with IANA. This specification updates the process to enable organisation prefixes to be registered allowing organisations to manage their own namespace within a certain set of rules.

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 October 21, 2017.

Copyright Notice

Copyright (c) 2017 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. Requirements Language	2
2. Introduction	2
3. Organisation Prefix	4
4. Security Considerations	5
5. IANA Considerations	5
5.1. Constrained RESTful Environments (CoRE) Parameters Registry Update	5
6. Acknowledgements	6
7. Changelog	6
8. References	7
8.1. Normative References	7
8.2. Informative References	7
8.3. URIs	7
Authors' Addresses	7

1. Requirements Language

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

2. Introduction

[RFC6690] "Constrained RESTful Environments (CoRE) Link Format" defines the Resource Type 'rt' and Interface Description 'if' link attributes. In order to co-ordinate the use of these attributes, sections 7.4 and 7.5/[RFC6690] establish IANA registries to register link attribute values for 'rt' and 'if'.

In order to register a new 'rt' and 'if' link attribute value the [RFC5226] "Specification Required" process is followed for each value. As part of the process a designated expert will examine the specification to enforce a number of requirements including:

- o Registration values MUST be related to the intended purpose of these attributes as described in Section 3/[RFC6690].
- o Registered values MUST conform to the ABNF reg-rel-type definition of Section 2/[RFC6690], meaning that the value starts with a lowercase alphabetic character, followed by a sequence of

lowercase alphabetic, numeric, ".", or "-" characters, and contains no white space.

- o It is recommended that the period "." character be used for dividing name segments and that the dash "-" character be used for making a segment more readable. Example Interface Description values might be "core.batch" and "core.link-batch".
- o URIs are reserved for free use as extension values for these attributes and MUST NOT be registered.

The IANA CoRE resource type and interface description registry can be found at: IANA CoRE Registry [1].

Given the scope of the Internet of Things (IoT) the potential number of resource types (and to a lesser extent interface types) is potentially quite large. This would lead to a large number of requests for designated expert review. It would also mean additional work for the IANA to process each request.

The current trend for the definition of resource types and interface descriptions is that a few standards organisations have defined a large number of values.

For example the "OIC Resource Type Specification v1.1.0" [OICResSpec] contains 64 resource types.

ETSI oneM2M also defines a large number of resource types. For example is "Home Appliances Information Model and Mapping" [oneM2MTS0023].

The Open Mobile Alliance (OMA) also make use of resources types.

The above three organisations also have their own registry and procedures for adding resource types. Trying to keep the IANA registry aligned with the individual organisation registries would also add additional burden.

A significant amount of work could be saved by allowing organisations to register a prefix under which they can administer their own resources negating the need for the IANA and the designated IANA expert to be involved for each resource registration.

There were discussions at the IETF#97 meeting in Seoul about how to tackle this issue. There were broadly two camps in the discussion:

- 1 - Prefixes are a bad idea as they discourage people from resource re-use and create little "kingdoms".

2 - Prefixes are OK. They've been used before and people will coalesce on a common set eventually.

Clearly some sort of middle ground is needed to move forward.

Clearly resource re-use is a valid goal. In order for this to occur organisations/people requesting a new resource type would need to consider the existing resource types and see if it is applicable to them. Presumably if they can't use an existing type then they must have a reason why? One approach could be to introduce a new step into the registration process where the requester must specify any similar resources and why they cannot be used. This does of course add extra burden on the requester to document it and extra burden on the expert to evaluate it. Then there is the issue of what suffices for the analysis and what are the criteria for the expert to accept it? This may not result in reduced registrations but instead create more workload for registrations.

Currently all the rt registrations have been from standards organisations not individuals. The process for registration needs to be simple enough that people/companies have an incentive to register than rather than simply use and squat on a name without registering it.

It does have to be noted that today there is nothing stopping people/organisations from duplicating resources in their registration. An organisational prefix would not make this worse.

Editor's note: Interface descriptions should be considered

This specification updates the [RFC6690] IANA registration procedures to allow the possibility to register a pre-fix.

3. Organisation Prefix

As indicated by [RFC6690] registered values MUST conform to the ABNF reg-rel-type definition, meaning that the value starts with a lowercase alphabetic character. Therefore an organisation registering a prefix MUST register a lowercase alphabetic sequence of characters. It MUST be followed by a ".".

For example: "foo."

This will allocate the namespace "foo." to the organisation. The organisation will then be responsible for maintaining resources within this name space.

E.g. "foo.sensor", "foo.actuator" could be allocated without requiring registration with IANA.

4. Security Considerations

This specification updates the [RFC6690] IANA Considerations. No additional protocol security impacts to what is already described in [RFC6690] are foreseen.

The use of organisational prefixes introducing the possibility that people request prefixes for an organisation that they do not represent. The IANA considerations in this specification require that the designated expert determine if the person requesting a prefix represents the organisation related to the prefix.

5. IANA Considerations

5.1. Constrained RESTful Environments (CoRE) Parameters Registry Update

This specification updates the Constrained RESTful Environments (CoRE) Parameter Registry by allowing the registration of an organisation prefix for Resource Type (rt=) and Interface Description (if=) Link Target Attribute values.

Organisation prefixes are registered by using the Specification Required policy (see [RFC5226], which requires review by a designated expert appointed by the IESG or their delegate.

The designated expert will enforce the following requirements:

- o The registered prefix MUST conform to the ABNF reg-rel-type definition of Section 2/[RFC6690], meaning that the value starts with a lowercase alphabetic character followed by a period ".".
- o The registered prefixes are assigned on a first come first served basis.
- o Prefixes must be requested by a representative of the organisation applying for the prefix and must be representative of the organisation. E.g. organisation "foo" trying to register "ietf." would not be representative.

The specification MUST:

- o Specify the procedures for registering values within the prefixed namespace. It ideally SHOULD provide a link where current and future registered values may be found.

- o Indicate that registered values within the prefixed namespace MUST conform to the ABNF `reg-rel-type` definition of Section 2/[RFC6690]. This means that the prefix MUST be followed by a sequence of lowercase alphabetic, numeric, ".", or "-" characters, and contains no white space. Note: It is not recommended to immediately follow the prefix with an additional period ".", e.g. "foo..".
- o Use the recommendation that the period "." character be used for dividing name segments and that the dash "-" character be used for making a segment more readable. Example Interface Description values might be "core.batch" and "core.link-batch".

Registration requests consist of the completed registration template below, with the reference pointing to the required specification. To allow for the allocation of values prior to publication, the designated expert may approve registration once they are satisfied that a specification will be published.

The registration template for both sub-registries is:

- o Prefix Value:
- o Description:
- o Reference:
- o Notes: [optional]

Registration requests should be sent to the `core-parameters@ietf.org` mailing list, marked clearly in the subject line (e.g., "NEW RESOURCE TYPE PREFIX - example" to register an "example" relation type or "NEW INTERFACE DESCRIPTION PREFIX - example" to register an "example" Interface Description).

Handling and the decision process is as per section 7.4/[RFC6690].

6. Acknowledgements

TBD

7. Changelog

draft-groves-core-rfc6690up-01:

- o Keepalive update. No changes.

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, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.

8.2. Informative References

- [OICResSpec] "OIC Resource Type Specification v1.1.0", 2016, <<https://openconnectivity.org/resources/specifications/draft-candidate-specifications>>.
- [oneM2MTS0023] "TS 0023 v2.0.0 Home Appliances Information Model and Mapping", 2015, <<http://www.onem2m.org/technical/published-documents>>.

8.3. URIs

- [1] <http://www.iana.org/assignments/core-parameters/core-parameters.xhtml>

Authors' Addresses

Christian Groves
Australia

Email: cngroves.std@gmail.com

Weiwei Yang
Huawei
P.R.China

Email: tommy@huawei.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: October 21, 2017

C. Groves

W. Yang
Huawei
April 19, 2017

SenML Base Time Offset Attribute
draft-groves-core-senml-bto-01

Abstract

SenML [I-D.ietf-core-senml] defines a base time attribute and time value which is used to determine the time when a value is recorded. In some applications a SenML pack will contain a series of records related to a constant sample time interval, e.g. once every 60 seconds. This means that the time attribute will be required for each record. This document defines a new "time offset" base attribute that allows a sender to include the time for the sample interval between records. If the "time offset" base attribute is used the sender will not send the time attribute for each record, minimising message and storage size.

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <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 October 21, 2017.

Copyright Notice

Copyright (c) 2017 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	4
3. SenML Structure and Semantics	4
3.1. Base Attributes	4
3.2. Regular Attributes	4
3.3. Considerations	4
4. JSON Representation (application/senml+json)	5
5. CBOR Representation (application/senml+cbor)	5
6. XML representation (application/senml+xml)	5
7. EXI Representation (application/senml-exi)	6
8. Security Considerations	7
9. IANA Considerations	7
10. Acknowledgements	8
11. Changelog	8
12. Normative References	8
Authors' Addresses	8

1. Introduction

SenML currently defines the base time "bt" and time "t" attributes to indicate the time that each value in a SenML record is recorded. This means that for each record (value "v") that a "t" attribute is required. The example from section 5.1.2/[I-D.ietf-core-senml] is copied below to illustrate a SenML pack with multiple records.

```
[ {"bn": "urn:dev:ow:10e2073a01080063",
  "bt": 1320067464,
  "bu": "%RH",
  "v": 21.2, "t": 0 },
  { "v": 21.3, "t": 10 },
  { "v": 21.4, "t": 20 },
  { "v": 21.4, "t": 30 },
  { "v": 21.5, "t": 40 },
  { "v": 21.5, "t": 50 },
  { "v": 21.5, "t": 60 },
  { "v": 21.6, "t": 70 },
  { "v": 21.7, "t": 80 },
  { "v": 21.5, "t": 90 },
  ...
```

Figure 1: SenML pack with multiple records

As can be seen in the example above there is a fixed offset between the data points of 10 seconds.

This document proposes a base time offset "bto" attribute that is used to indicate the offset between datapoints when a fixed offset is used. This negates the need to include the "t" attribute for each data point. The example below takes the above example and applies the base time offset to it.

```
[ {"bn": "urn:dev:ow:10e2073a01080063",
  "bt": 1320067464,
  "bto": 10,
  "bu": "%RH",
  "v": 21.2},
  { "v": 21.3},
  { "v": 21.4},
  { "v": 21.4},
  { "v": 21.5},
  { "v": 21.5},
  { "v": 21.5},
  { "v": 21.6},
  { "v": 21.7},
  { "v": 21.5},
  ...
```

Figure 2: SenML pack with multiple records using base time offset

It can be seen that it results in a record and overall pack size reduction. The receiver of the record would use the base time "bt" for the initial data point, e.g. "v": 21.2 would have a timestamp of

1320067464. Each subsequent record would have a time stamp equal to the previous record plus the base time offset "bto", e.g. "v": 21.3 would have a timestamp of 1320067474, "v": 21.4 would have a timestamp of 1320067484, and so on.

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].

See [I-D.ietf-core-senml] for further definitions.

3. SenML Structure and Semantics

3.1. Base Attributes

This document adds an additional "base time offset" attribute.

Base Time Offset: The base time offset represents a fixed time offset between each record in a SenML pack. The first record represents time zero (or the base time if provided) and the offset is then applied to each subsequent record. Either a positive or negative base time offset is allowed.

3.2. Regular Attributes

This document modifies the behaviour of the time attribute.

Time: If the base time offset attribute is used in a SenML pack then the time attribute shall not be used in the individual records.

Editor's note: One theoretical use case may be to include the time attribute if the entry's time deviates from the regular offset. It is not seen that such a corner case warrants the extra complexity._

3.3. Considerations

To simplify implementation, the use of base time offset is limited to:

- o Fixed time increasing or decreasing records from the time stamp associated with the first record in the SenML pack. This means that usages such as described by the 2nd example in 5.1.2/[I-D.ietf-core-senml] where negative time offset from time zero being provided by the last record in a SenML pack are not possible.

Editor's note: It could be possible to facilitate this use case by allowing a t=0 to be set on the last record with the use of a negative base time offset value. However again it's not seen that such a corner case warrants the extra complexity in having to store values and calculate offsets at the end of transmission/reception.

- o It is not possible to for two records in the SenML pack to have the same time. For example the inclusion of a record for a voltage measurement followed by a current measurement would result in the records having times with a difference of the time offset.

4. JSON Representation (application/senml+json)

This document defines an additional SenML label (JSON object member name) as shown in Table 1 below.

Name	label	Type
Base Time Offset	bto	Numer

Table 1: JSON SenML Labels

5. CBOR Representation (application/senml+cbor)

As per section 6/[I-D.ietf-core-senml], for CBOR the string map key "bto" shall be used to indicate the use of the base time offset.

6. XML representation (application/senml+xml)

This document defines an addition XML attribute as shown in Table 2 below.

Name	XML	Type
Base Time Offset	bto	double

Table 2: XML SenML Labels

The RelaxNG schema for the XML is:

```
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 bto { xsd:double }?,

  attribute l { xsd:string }?,

  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
```

7. EXI Representation (application/senml-exi)

As per clause 8/[I-D.ietf-core-senml] extensions are indicated through the use of an EXI schemaID options. The EXI schemaID options MUST be set to the value of "b" indicating the scheme provided in this specification.

The following is the XSD Schema to be used for strict schema guided EXI processing.

```

<?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="bu" type="xs:string" />
        <xs:attribute name="bver" type="xs:int" />
        <xs:attribute name="bto" type="xs:double" />
        <xs:attribute name="l" type="xs:string" />
        <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>

```

8. Security Considerations

No extra security issues are seen beyond those described in [I-D.ietf-core-senml].

9. IANA Considerations

This document proposes one new entry in the IANA SenML label registry.

Label Name: Base Time Offset

Label: bto

CBOR: -

XML Type: Double

ID: b

Reference: This specification.

10. Acknowledgements

TBD

11. Changelog

draft-groves-core-senml-bto-01

- o General: Changed "SenML Package" to "SenML Pack"

12. Normative References

[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-05 (work in progress), March 2017.

[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>>.

Authors' Addresses

Christian Groves
Australia

Email: cngroves.std@gmail.com

Weiwei Yang
Huawei
P.R.China

Email: tommy@huawei.com

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 11, 2017

C. Groves
W. Yang
Huawei
March 10, 2017

SenML Options
draft-groves-core-senml-options-00

Abstract

SenML [I-D.ietf-core-senml] defines an initial set of base and regular attributes which are tied to a particular version of SenML. SenML also allows the definition of additional attributes by extending the syntax with a new label. Allowing the extension of attributes brings the problem of how do endpoints negotiate whether the new attribute can be used or not? This document discusses the issue and proposes some potential solutions to this issue.

Status of This Memo

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

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

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

This Internet-Draft will expire on September 11, 2017.

Copyright Notice

Copyright (c) 2017 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. Solution space analysis	3
3.1. Version Numbering	4
3.2. Mandatory / Optional Indication	4
3.3. Options Mechanism	5
3.4. Media Type Definition and Parameters	6
4. Solution Proposal	7
4.1. Accept Media Type Parameter (AMTP) Option	9
4.2. Content-Format Media-Type Parameter Option	10
5. Security Considerations	11
6. IANA Considerations	11
7. Acknowledgements	11
8. Changelog	11
9. References	11
9.1. Normative References	11
9.2. Informative References	12
Authors' Addresses	13

1. Introduction

SenML [I-D.ietf-core-senml] defines an initial set of base and regular attributes which are tied to a particular version of SenML. SenML also allows the definition of additional attributes by extending the defined syntax with a new label. Allowing the extension of attributes brings the problem of how do endpoints negotiate whether the new attribute can be used or not?

For example: A CoAP client issues a GET that indicates support of SenML through the use of an CoAP Accept option. A CoAP server supports the SenML attributes defined in [I-D.ietf-core-senml] and in addition supports the Base Time Offset (BTO) [I-D.groves-core-senml-bto] attribute. The server responds using the BTO attribute.

```
[ {"bn": "urn:dev:ow:10e2073a01080063",  
  "bt": 1320067464,  
  "bto": 10,  
  "bu": "%RH",  
  "v": 21.2},  
  { "v": 21.3},  
  { "v": 21.4},  
  { "v": 21.4},  
  { "v": 21.5},  
  { "v": 21.5},  
  { "v": 21.5},  
  { "v": 21.6},  
  { "v": 21.7},  
  { "v": 21.5},  
  ...
```

Figure 1: Response with SenML using base time offset

As the CoAP client does not understand the "bto" attribute it will ignore the attribute. This means that the time information is lost for each of the SenML records. Whereas if the Server had not used "bto" the client would have been able to understand the information.

This is mainly a problem when the server provides a response to a message (i.e. GET) rather than when a client uses the SenML media type in a message (i.e. PUT). In this later case the client can modify its behaviour and not use an attribute based on an error response from the server.

A solution is needed to prevent incompatible attributes from being used.

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].

See [I-D.ietf-core-senml] for further definitions.

3. Solution space analysis

The extension of protocols in a compatible manner is not a new problem. Some approaches to handle the are discussed below. The discussion below is not meant to be treatise on protocol extension but to highlight potential solutions.

3.1. Version Numbering

A version number is used to indicate when changes have been made to a syntax. In some protocols a major version is used to indicate non-backward compatible changes and a minor version is used to indicate backwards compatible changes. SenML does use a version number however it is an integer (no major and minor). It is used to indicate the version number of the media type format. The version does not appear in the media type format name. However presumably the proposed registrations (e.g. senml+json) imply the use of version number 10 (see sect.4.1/[I-D.ietf-core-senml]). If in the future there is a new version how does the endpoint negotiate which SenML version to use? A simple solution would be to create a new media type name incorporating the new version e.g. "senml_v11+json". A CoAP endpoint could then use an Accept option to indicate support for "senml+json" and/or "senml_v11+json".

This method does generally mean that for each new attribute and version the endpoints must at least understand all previously defined attributes. Although major versions in some protocols do not maintain backwards compatibility.

SenML does indicate that for certain representations i.e. EXI representation (application/senml+exi) that the schemaID number must be updated when the syntax is updated for a new attribute. This is effectively a version mechanism but the other representation formats do not follow this approach.

However the current SenML draft does allow the definition of additional attributes without increasing the SenML version number. Indeed there is a trend at the IETF that protocol versions change very rarely. Instead updates are incorporated via option or extension mechanisms.

Therefore it seems that an approach of utilising a version number for each additional new attribute does not seem appropriate for SenML.

3.2. Mandatory / Optional Indication

Some protocols use a mechanism to indicate whether a parameter is mandatory or optional to understand. This is based on the assumption that a sending endpoint knows the functions that a parameter/s relate to and can indicate whether the receiving endpoint must understand the parameter/s to implement the function. A receiving endpoint will analyse the received parameters and if it does not understand the parameter it will check the mandatory/option tag to see what it should do. If the parameter is marked as mandatory then the receiving endpoint will generate an error. If the parameter is

marked as optional then the receiving endpoint will continue processing in knowledge that it doesn't need the parameter.

CoAP uses a mechanism similar to this for Options. By looking at the option number an endpoint can determine whether it is critical or not.

SenML does indicate that the version indicates the mandatory to understand attributes (sect. 4.3/[I-D.ietf-core-senml]). SenML also indicates that some attributes (i.e. base attributes) are optional but this is in context of "optional to be used" rather than "optional to be understood".

Whilst a method could be defined to indicate in SenML whether an attribute is mandatory or optional, its not clear that it would be useful. Given the number of use cases where CoAP can be used a server may not know which information in a SenML pack is relevant for a client. E.g. Whilst a server may return time information associated with a record it doesn't actually know whether it is useful to the client. The usefulness is application specific to the client.

Therefore it seems this approach is also not appropriate for SenML.

3.3. Options Mechanism

Some protocols allow the optional parts of the protocol to be negotiated during the initial protocol negotiation. For example the SIP protocol has the OPTIONS method [RFC3261]. The CLUE protocol [I-D.ietf-clue-protocol] also defines an extension method where an OPTIONS message is used to negotiate the protocol extensions. The benefits of such an approach is that two endpoints can negotiate which extensions they will use in a session ensuring compatible communications.

However these approaches assume an application level session where there are establishment, communication and release phases. SenML is a media type format primarily defined to be used with the HTTP and CoAP protocols. These protocols don't follow a session based approach.

HTTP/1.1 does have the OPTIONS method [RFC2616] however the use is largely undocumented. CoAP does not have an equivalent method.

CoAP does have a method for negotiating signalling through "Signaling Option Numbers" (sect.4.2/[I-D.ietf-core-coap-tcp-tls]). This however is more used to negotiate the properties of the signalling

connection than any elements of the CoAP payload (not withstanding the Blockwise transfer).

CoAP does have the OPTIONS mechanism allowing for the definition of optionality functionality associated with a CoAP message. It also defines the concept of critical and elective options. Two options related to Content-type/Content-format are "Content-Format" and "Accept". These options allow endpoints to indicate the media types are using or wish to use. HTTP also uses these options.

CoAP also allows a per message exchange of what is supported for a particular resource. This seems more appropriate than a protocol level negotiation of the support of SenML attributes.

This functionality is very close to what needs to be achieved for negotiating SenML attributes.

3.4. Media Type Definition and Parameters

Potentially the media type name could be used to indicate versions or extensions. This may be appropriate where there are seldom changes that affect the whole media type. However it may become unwieldy if the media type name is used to define combinations of SenML attributes, e.g. given 3 extensions a, b and c you could end up with media names / content formats for a, b, c, a+b, a+c, b+c, a+b+c. The problem gets worse each time an extension is added. It is made even worse because Table 7/[I-D.ietf-core-senml] defines 8 different content formats for SenML that would need to be updated. To allow combinations of these parameters on the media types defined in SenML it would need 56 Content-format code points. The content format range 0..255 for IETF specifications isn't particularly large.

[RFC6838] allows for the registration of media type parameters. This allows further companion information to be included along with the media type. Charset is a common parameter (See [RFC3023]). This information could be used to provide version or option information associated with a media type.

This appears to be a good solution to indicate if additional SenML attributes are supported in a media type. Whilst HTTP supports media type parameters, CoAP does not support media type parameters or extensions (i.e. see sect.10.2.2/[RFC7252]). Meaning that parameters cannot be used as a common approach for HTTP and CoAP. However this solution is used in section 16.9.1/[I-D.ietf-cose-msg] which takes the approach of defining an optional parameter for the "application/cose". It then assigns multiple CoAP Content-Formats for the values associated with the optional parameter (see sect.16.10/[I-D.ietf-cose-msg]).

4. Solution Proposal

There doesn't appear to be one outstanding approach for negotiating SenML attributes common to both HTTP and CoAP. The authors believe that a hybrid approach as per [I-D.ietf-cose-msg] is needed in order to be able to negotiate which SenML extension attributes are used.

The first part would be the definition of a optional media-type parameter that allows an endpoint utilising HTTP to indicate the SenML extension attributes that it is using or accepts. This could be in the form of a comma separated string list of SenML labels from those registered in the SenML label registry. Only attributes NOT defined in [I-D.ietf-core-senml] would be allowed.

e.g. Content-type: application/senml+json; ext=abc,xyz

In order to allow this functionality in the base version of SenML an optional parameter would be needed in the media type registrations in sect.11.3/[I-D.ietf-core-senml].

i.e. o Optional parameter: SenML extensions

This parameter indicates which SenML extensions are associated with the media type. The parameter is defined by the following ABNF:

```
SenML-ext = "ext" "=" <"> senml-label *("," senml-label) <">
; Note: this follows the {{RFC2616}} quoted-string form.
; senml-label is the label string from the list of IANA
; registered SenML labels.
; Only non-{{I-D.ietf-core-senml}} labels are allowed.
```

For example: ext="a,b,c";

If the group decides that there will only ever be a small number of SenML extensions then the simplest approach would be to follow [I-D.ietf-cose-msg] and define multiple CoAP content formats associated with potential extensions. This would be done in which ever document defines the SenML extension. For example [I-D.groves-core-senml-bto] would add the following to the IANA considerations section:

Media Type	Encoding	ID	Reference
application/senml+json; ext="bto"		TBD	TBD
application/sensml+json; ext="bto"		TBD	TBD
application/senml+cbor; ext="bto"		TBD	TBD
application/sensml+cbor; ext="bto"		TBD	TBD
application/senml+xml; ext="bto"		TBD	TBD
application/sensml+xml; ext="bto"		TBD	TBD
application/senml+exi; ext="bto"		TBD	TBD
application/sensml+exi; ext="bto"		TBD	TBD

Table 1: New CoAP content formats

Text would also need to be added to [I-D.ietf-core-senml] to describe the procedure for support of the media type parameter in CoAP.

If issuing potentially a large number of content-format numbers is problematic a separate approach could be taken. A new CoAP option could be defined to allow media-type parameters to be carried in CoAP messages when then Content-Format or Accept options are used. As the Content-Format and Accept options may be used in the same request (with two different media types) two new options would be required, one for the media-type parameters associated with the Content-Format Option and one for the media-type parameters associated with the Accept option.

Editor's note: Alternatives could include defining the option for both CoAP and HTTP. However this would likely mean that the option would become specific to SenML extensions rather than a general mechanism for carrying media type parameters.

As CoAP only allows a single Content-Format to be carried in the Content-Format and Accept options it would be straight forward to define an option that allows media-type parameters to be carried. One complication is that the encoding and syntax of the media-type parameters is up to the media-type definition. It could be a string, integer, binary, etc. Therefore the option value would need to be an opaque sequence of bytes. If the scope was limited to SenML then the option format would be a narrowed to a string of labels.

As multiple parameters could be defined for a media-type the mechanism must allow multiple media type parameter to be signalled in a CLUE message. One possible method is to define the option value syntax to allow multiple parameter to be specified as a single parameter value. Alternatively multiple instances of the option could be used in the CoAP message. This method is indicated in the IANA registration by allowing the option multiple times.

If a new generic option is defined it's not clear that [I-D.ietf-core-senml] would be the best place to define the option. If the scope is limited then [I-D.ietf-core-senml] would be appropriate. Whichever draft defines the options it would need to define them for registration with IANA along the lines of:

Number	Name	Reference
TBD	AMTP	TBD
TBD	CFMTP	TBD

Table 2: New CoAP Option Numbers

4.1. Accept Media Type Parameter (AMTP) Option

o The meaning of the option in a request: It indicates the media-type parameters associated with the content-format (media-type) specified in the Accept option.

Note: Some content-formats contain media-type parameters as part of the content-format ID registrations. The AMTP option SHALL not be used with these CoAP content-formats.

o The meaning of the option in a response: Not used.

o Whether the option is critical or elective: Critical as per the Accept option.

o Whether the option is Safe-to-Forward: Safe as per the Accept option.

Note: Potentially it could be unsafe to forward an opaque byte sequence that the proxy does not understand. However processing this option should only be done within the context of the media-type specified by the Accept option.

- o The format and length of the option's value: A variable length opaque sequence of bytes. The encoding of the bytes is defined as per the syntax for the parameters in the media-type definition document.
- o Whether the option must occur at most once or whether it can occur multiple times: Multiple times. Each instance containing a separate media-type parameter. Whether the option can be included multiple times for the one media type parameter is dependent on whether the media-type definition allows for multiple instances of the one media type parameter.
- o Default value: None unless the media-type indicated in the accept option defines a default parameter/s value.

4.2. Content-Format Media-Type Parameter Option

- o The meaning of the option in a request: When used together with the Content-Format option it indicates the media-type parameters associated with the content-format (media-type) specified in the content-format option.

Note: Some content-formats contain media-type parameters as part of the content-format ID registrations. The CFMTP option SHALL not be used with these CoAP content-formats.

- o The meaning of the option in a response: When used together with the Content-Format option it indicates the media-type parameters associated with the content-format (media-type) specified in the content-format option.

- o Whether the option is critical or elective: As per the Content-Format option it is elective.

- o Whether the option is Safe-to-Forward: Safe as per the Content-format option.

Note: Potentially it could be unsafe to forward an opaque byte sequence that the proxy does not understand. However processing this option should only be done within the context of the media-type specified by the Content-Format options.

- o The format and length of the option's value: A variable length opaque sequence of bytes. The encoding of the bytes is defined as per the syntax for the parameters in the media-type definition document.

- o Whether the option must occur at most once or whether it can occur multiple times: Multiple times. Each instance containing a separate media-type parameter. Whether the option can be included multiple times for the one media type parameter is dependent on whether the media-type definition allows for multiple instances of the one media type parameter.

- o Default value: None unless the media-type indicated in the content-format option defines a default parameter/s value.

5. Security Considerations

SenML security issues are described in [I-D.ietf-core-senml]. Some extra considerations are indicated above.

6. IANA Considerations

Section 4 discusses potential IANA registrations.

7. Acknowledgements

TBD

8. Changelog

TBD

9. References

9.1. Normative References

[I-D.groves-core-senml-bto]

Groves, C. and W. Yang, "SenML Base Time Offset Attribute", draft-groves-core-senml-bto-00 (work in progress), October 2016.

[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-04 (work in progress), October 2016.

[I-D.ietf-cose-msg]

Schaad, J., "CBOR Object Signing and Encryption (COSE)", draft-ietf-cose-msg-24 (work in progress), November 2016.

- [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>>.
- [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, <<http://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, <<http://www.rfc-editor.org/info/rfc3261>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<http://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, <<http://www.rfc-editor.org/info/rfc7252>>.

9.2. Informative References

- [I-D.ietf-clue-protocol] Presta, R. and S. Romano, "CLUE protocol", draft-ietf-clue-protocol-13 (work in progress), February 2017.
- [I-D.ietf-core-coap-tcp-tls] Bormann, C., Lemay, S., Tschafenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-07 (work in progress), March 2017.
- [RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media Types", RFC 3023, DOI 10.17487/RFC3023, January 2001, <<http://www.rfc-editor.org/info/rfc3023>>.

Authors' Addresses

Christian Groves
Huawei
Australia

Email: Christian.Groves@mail01.huawei.com

Weiwei Yang
Huawei
P.R.China

Email: tommy@huawei.com

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

K. Hartke
Ericsson
October 22, 2018

CoRE Applications
draft-hartke-core-apps-08

Abstract

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

Note to Readers

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

Status of This Memo

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

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

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

This Internet-Draft will expire on April 25, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

1. Introduction	2
2. CoRE Applications	3
2.1. Communication Protocols	4
2.1.1. URI Schemes	4
2.2. Representation Formats	5
2.2.1. Internet Media Types	5
2.3. Links	7
2.3.1. Link Relation Types	8
2.3.2. Template Variable Names	8
2.4. Forms	8
2.4.1. Form Relation Types	9
2.4.2. Form Field Names	9
2.5. Well-Known Locations	10
3. CoRE Application Descriptions	10
3.1. Template	11
4. URI Design Considerations	12
5. Security Considerations	14
6. IANA Considerations	14
7. References	14
7.1. Normative References	14
7.2. Informative References	15
Acknowledgements	16
Author's Address	17

1. Introduction

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

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

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

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

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

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

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

2. CoRE Applications

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

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

The API is generally comprised of:

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

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

2.1. Communication Protocols

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

2.1.1. URI Schemes

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

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

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

2.2. Representation Formats

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

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

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

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

2.2.1. Internet Media Types

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

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

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

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

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

Media types can be further refined by

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

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

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

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

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

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

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

2.3. Links

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

A link is comprised of:

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

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

There are two special kinds of links:

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

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

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

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

2.3.1. Link Relation Types

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

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

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

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

2.3.2. Template Variable Names

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

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

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

2.4. Forms

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

A form is comprised of:

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

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

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

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

2.4.1. Form Relation Types

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

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

2.4.2. Form Field Names

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

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

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

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

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

2.5. Well-Known Locations

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

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

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

3. CoRE Application Descriptions

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

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

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

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

3.1. Template

The fields of the template are as follows:

Application name:

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

URI schemes:

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

Media types:

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

Link relation types:

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

Template variable names:

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

Form relation types:

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

Form field names:

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

Well-known locations:

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

Interoperability considerations:

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

Security considerations:

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

Contact:

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

Author/Change controller:

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

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

4. URI Design Considerations

URIs [1] are a cornerstone of RESTful applications. They enable uniform identification of resources via URI schemes [7] and are used every time a client interacts with a particular resource or when a resource representation references another resource.

URIs often include structured application data in the path and query components, such as paths in a filesystem or keys in a database. It is common for many RESTful applications to use these structures not only as an implementation detail but also make them part of the

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

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

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

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

A good RESTful URI is:

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

5. Security Considerations

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

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

6. IANA Considerations

This document has no IANA actions.

7. References

7.1. Normative References

- [1] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [2] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<https://www.rfc-editor.org/info/rfc6838>>.
- [3] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [4] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/info/rfc5785>>.
- [5] Nottingham, M., "URI Design and Ownership", BCP 190, RFC 7320, DOI 10.17487/RFC7320, July 2014, <<https://www.rfc-editor.org/info/rfc7320>>.
- [6] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

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

7.2. Informative References

- [8] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [9] Berners-Lee, T., "Cool URIs don't change", 1998, <<http://www.w3.org/Provider/Style/URI.html>>.
- [10] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [11] Bray, T., Hollander, D., Layman, A., Tobin, R., and H. Thompson, "Namespaces in XML 1.0 (Third Edition)", World Wide Web Consortium Recommendation REC-xml-names-20091208, December 2009, <<http://www.w3.org/TR/2009/REC-xml-names-20091208>>.
- [12] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <<http://www.w3.org/TR/2008/REC-xml-20081126>>.
- [13] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., Yergeau, F., and J. Cowan, "Extensible Markup Language (XML) 1.1 (Second Edition)", World Wide Web Consortium Recommendation REC-xml11-20060816, August 2006, <<http://www.w3.org/TR/2006/REC-xml11-20060816>>.
- [14] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [15] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.

- [16] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", Ph.D. Dissertation, University of California, Irvine, 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf>.
- [17] Fielding, R., "REST APIs must be hypertext-driven", October 2008, <<http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven>>.
- [18] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <<https://www.rfc-editor.org/info/rfc7301>>.
- [19] Hammer-Lahav, E., Ed. and B. Cook, "Web Host Metadata", RFC 6415, DOI 10.17487/RFC6415, October 2011, <<https://www.rfc-editor.org/info/rfc6415>>.
- [20] Masinter, L., "MIME and the Web", draft-masinter-mime-web-info-02 (work in progress), January 2011.
- [21] Richardson, L. and M. Amundsen, "RESTful Web APIs", O'Reilly Media, ISBN 978-1-4493-5806-8, September 2013.
- [22] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [23] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [24] Wilde, E., "The 'profile' Link Relation Type", RFC 6906, DOI 10.17487/RFC6906, March 2013, <<https://www.rfc-editor.org/info/rfc6906>>.

Acknowledgements

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

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

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

This work was funded in part by Nokia.

Author's Address

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 30, 2018

G. Selander
F. Palombini
Ericsson AB
K. Hartke
Universitaet Bremen TZI
July 29, 2017

Requirements for CoAP End-To-End Security
draft-hartke-core-e2e-security-reqs-03

Abstract

This document analyses threats to CoAP message exchanges traversing proxies and derives security requirements for mitigating those threats.

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 30, 2018.

Copyright Notice

Copyright (c) 2017 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
1.1. Assets and Scope	4
1.2. Terminology	5
2. Proxying	6
2.1. Threats and Security Requirements	7
2.1.1. Client-side	7
2.1.1.1. Threat 1: Spoofing	8
2.1.1.2. Threat 2: Delaying	9
2.1.1.3. Threat 3: Withholding	9
2.1.1.4. Threat 4: Flooding	9
2.1.1.5. Threat 5: Eavesdropping	9
2.1.1.6. Threat 6: Traffic Analysis	9
2.1.2. Server-side	11
2.1.2.1. Threat 1: Spoofing	12
2.1.2.2. Threat 2: Delaying	12
2.1.2.3. Threat 3: Withholding	12
2.1.2.4. Threat 4: Flooding	12
2.1.2.5. Threat 5: Eavesdropping	13
2.1.2.6. Threat 6: Traffic Analysis	13
2.2. Solutions	14
2.2.1. Forwarding	15
2.2.1.1. Examples	15
2.2.1.2. Functional Requirements	17
2.2.1.3. Processing Rules	17
2.2.1.4. Authenticity	17
2.2.1.5. Confidentiality	19
2.2.2. Caching	19
2.2.2.1. Examples	19
2.2.2.2. Functional Requirements	21
2.2.2.3. Processing Rules	21
2.2.2.4. Authenticity	22
2.2.2.5. Confidentiality	23
3. Publish-Subscribe	24
3.1. Threats and Security Requirements	24
3.1.1. Subscriber-side	24
3.1.1.1. Threat 1: Spoofing	26
3.1.1.2. Threat 2: Delaying	27
3.1.1.3. Threat 3: Withholding	27
3.1.1.4. Threat 4: Flooding	27
3.1.1.5. Threat 5: Eavesdropping	27
3.1.1.6. Threat 6: Traffic Analysis	27
3.1.2. Publisher-side	27
3.2. Solutions	28
3.2.1. Brokering	28
3.2.1.1. Functional Requirements	30
3.2.1.2. Processing Rules	30

3.2.1.3. Authenticity	30
3.2.1.4. Confidentiality	30
4. Security Considerations	30
5. IANA Considerations	30
6. References	31
6.1. Normative References	31
6.2. Informative References	31
Acknowledgments	32
Authors' Addresses	32

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a Web application protocol designed for constrained nodes and networks [RFC7228]. CoAP makes use of Datagram Transport Layer Security (DTLS) [RFC6347] for security. At the same time, CoAP relies on proxies for scalability and efficiency. Proxies reduce response time and network bandwidth use by serving responses from a shared cache or enable clients to make requests that these otherwise could not make.

CoAP proxies need to perform a number of operations on requests and responses to fulfill their purpose, which requires the DTLS security associations to be terminated at each proxy. The proxies therefore do not only have access to the data required for performing the desired functionality, but are also able to eavesdrop on or manipulate any part of the CoAP payload and metadata exchanged between client and server, or inject new CoAP messages without being protected or detected by DTLS.

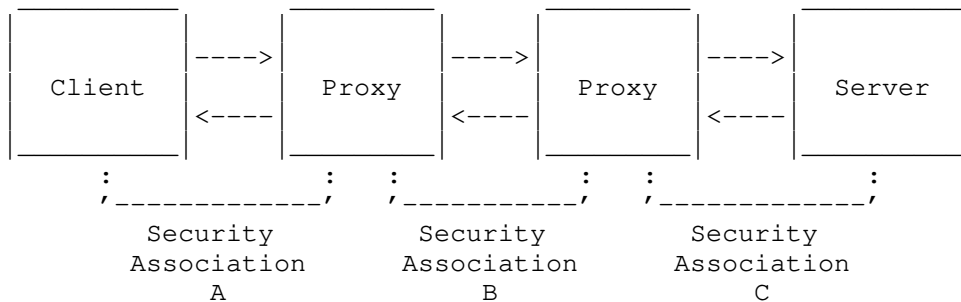


Figure 1: Hop-by-Hop Security

One way to mitigate this threat is to secure CoAP communication at the application layer using an object-based security mechanism such as CBOR Object Signing and Encryption (COSE) [RFC8152] instead of or in addition to the security mechanisms at the network layer or transport layer. Such a mechanism can provide "end-to-end security"

at the CoAP layer (Figure 2) in contrast to the "hop-by-hop security" that DTLS provides at the CoAP layer (Figure 1).

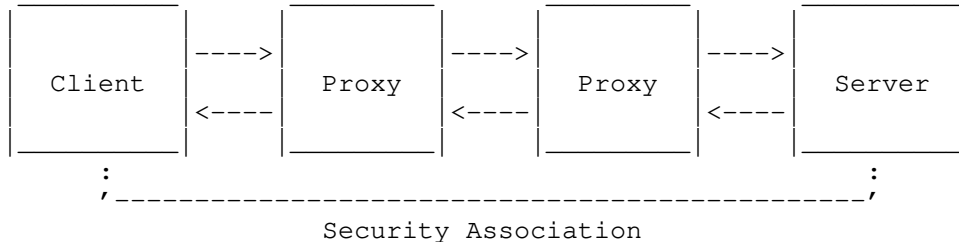


Figure 2: End-to-End Security

This document analyses security aspects of sensor and actuator communications over CoAP that involve proxies (Section 2) and publish-subscribe brokers (Section 3). The analysis is based on the identification of assets associated with these communications and considering the potential threats posed by proxies to these assets. The threat analysis provides the basis for deriving security requirements that a solution for CoAP end-to-end security should meet.

1.1. Assets and Scope

In general, there are the following assets that need to be protected:

- o The devices at the two ends and their (often very constrained) system resources such as available memory, storage, processing power and energy.
- o The physical environment of the devices fitted with sensors and actuators. Access to the physical environment is assumed to be provided through CoAP resources that allow a remote entity to retrieve information about the physical environment (such as the current temperature) or to produce an effect on the physical environment (such as the activation of a heater).
- o The communication infrastructure linking the two devices, which often contains some very constrained networks.
- o The data generated and stored in the involved devices.

An intermediary can directly interfere with the interactions between the two ends and thereby have an impact on all these assets. For example, flooding a device with messages has an impact on system resources, and the successful manipulation of an actuator command

(data generated by an involved device) can have a severe impact on the physical environment. An intermediary can also affect the communication infrastructure, e.g., by dropping messages.

Even if an intermediary is trustworthy, it may be an attractive target for an attack, since such nodes are aggregation points for message flows and may be an easier target from the Internet than the sensor and actuator nodes residing behind them. An intermediary may become subject to intrusion or be infected by malware and perform the attacks of a man-in-the-middle.

The focus of this document is on threats from intermediaries to interactions between two CoAP endpoints. Other types of threats, for example, attacks involving physical access to the CoAP-speaking devices, are out of scope of this document.

Since intermediaries may perform a service for the interacting endpoints, there is a trade-off between the intermediaries' desired functionality and the ability to mitigate threats to the endpoints executed through an intermediary.

1.2. Terminology

Readers are expected to be familiar with the terms and concepts described in [RFC7252] and [RFC7641].

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 key word "NOT REQUIRED" is interpreted as synonymous with the key word "OPTIONAL".

2. Proxying

To assess what impact various threats have to the assets, we need to specify and analyse how the proxies operate.

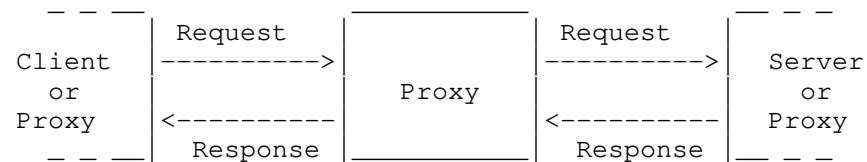


Figure 3: A Proxy

Generally speaking, the functionality of a proxy is to receive a request from a client and to send a response back to that client. There are two ways for the proxy to satisfy the request:

- o The proxy constructs and sends a request to the server indicated in the client's request, receives a response from that server and uses the received data to construct the response to the client.
- o The proxy uses cached data to construct the response to the client.

In both cases, the proxy needs to read some parts both of the request from the client and the response from the server to accomplish its task.

The following subsections analyse the threats posed by a proxy from the perspective of the client on the one hand (Section 2.1.1) and the perspective of the server on the other hand (Section 2.1.2). Section 2.2 then presents the design space for possible security solutions to mitigate the threats.

2.1. Threats and Security Requirements

2.1.1. Client-side

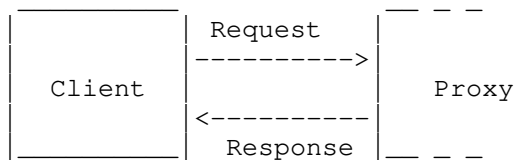


Figure 4: The Client End

The client sends a request to the proxy and waits for a response.

From the perspective of the client, there are three possible flows:

- o The client receives a response.
Reasons include:
 - * The proxy duly processed the request and returns a response based on data it obtained from the origin server.
 - * The proxy encountered an unexpected condition and returns an error response according to specification (e.g., 5.02 Bad Gateway or 5.04 Gateway Timeout).
 - * (Threat 1:) The proxy spoofs a response. For example, the proxy could return a stale or outdated response based on data it previously obtained from the server or some fourth party, or could craft an illicit response itself.
 - * (Threat 2:) The proxy duly processed the request but delays the return of the response.
- o The client does not receive a response.
Reasons include:
 - * The client times out too early.
 - * (Threat 3:) The proxy withholds the response.
- o The client receives too many responses.
Reasons include:
 - * (Threat 4:) The proxy floods the client with responses.

Furthermore, there are threats related to privacy:

- o (Threat 5:) The proxy eavesdrops on the data in the request from the client.
- o (Threat 6:) The proxy measures the size, frequency or distribution of requests from the client.

Note that "cache poisoning" -- the case of caching injected incorrect responses -- is covered from the point of view of the client: it may result in the client receiving a spoofed message or being flooded, or affect other nodes such that the client times out too early.

2.1.1.1. Threat 1: Spoofing

With one exception (see below), this threat is REQUIRED to be mitigated by the security solution: the client MUST verify that the response is an "authentic response" before processing it.

The definition of an "authentic response" depends on the desired proxy functionality and protection level (see Section 2.2), but usually means that the client can obtain proof for some or all of the following items:

- o that the requested action was executed by the origin server;
- o that the data originates from the origin server and has not been altered on the way;
- o that the data matches the specifications of the request (such as the target resource);
- o that the data is fresh (when the data is cacheable);
- o that the data is in sequence (when observing a resource).

The proof can, for example, involve a message authentication code that the proxy obtains from the origin server and includes in the response or an additional challenge-response roundtrip.

Exception: A CoAP proxy is specified to return an error response (such as 5.02 Bad Gateway or 5.04 Gateway Timeout) when it encounters an error condition. Since the condition occurs at the proxy and not at the origin server, the response will not be an "authentic response" according to the above definition. (A proxy cannot obtain a proof that the server is unreachable from an unreachable server.) Thus, a client cannot tell if the proxy sends the response according to specification or if it spoofs the response. This threat is NOT REQUIRED to be mitigated by the security solution.

2.1.1.2. Threat 2: Delaying

This threat is REQUIRED to be mitigated by the security solution. Delay attacks are important to mitigate in certain applications, e.g., when using CoAP with actuators. A detailed problem statement and candidate solution can be found in [I-D.mattsson-core-coap-actuators].

2.1.1.3. Threat 3: Withholding

This threat is NOT REQUIRED to be mitigated by the security solution, since a client cannot tell if the proxy does not send a response because it is hasn't received a response from the origin server yet or if it intentionally withholds the response.

2.1.1.4. Threat 4: Flooding

A CoAP client is specified to reject any response that it does not expect. This can happen before the client verifies whether the response is authentic. Therefore, a flood of responses is primarily a threat to the system resources of the client, in particular to its energy. This threat is NOT REQUIRED to be mitigated by the security solution, but a client SHOULD generally defend against flooding attacks.

2.1.1.5. Threat 5: Eavesdropping

This threat is REQUIRED to be mitigated by the security solution: clients MUST confidentiality protect the data in the requests they send.

Note that this requirement is in conflict with the requirement that the proxy needs to be able to read some parts of the requests in order to accomplish its task. Section 2.2 analyses which parts can be encrypted depending on the desired proxy functionality and protection level. In general, a security solution SHOULD confidentiality protect all data that is not needed by the proxy to accomplish its task.

The keys used for confidentiality protection MUST provide forward secrecy.

2.1.1.6. Threat 6: Traffic Analysis

This threat is NOT REQUIRED to be mitigated by the security solution.

It is RECOMMENDED that applications analyse the risks associated with application information leaking from the messages flow and assess the

feasibility to protect against various threats, e.g., by obfuscating parameters transported in plain text, aligning message flow and traffic between the different cases, adding padding so different messages become indistinguishable, etc.

2.1.2. Server-side

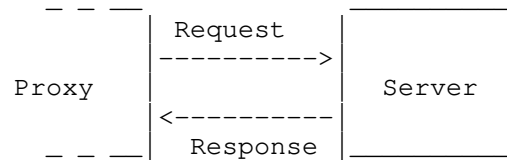


Figure 5: The Server End

A server listens for a request and returns a response.

From the perspective of the server, there are three possible flows:

- o The server receives a request.
Reasons include:
 - * The proxy makes a request on behalf of a client according to specification.
 - * The proxy makes a request (e.g., to validate cached data) on its own behalf.
 - * (Threat 1:) The proxy spoofs a request.
 - * (Threat 2:) The proxy sends a request with delay.
- o The server does not receive a request.
Reasons include:
 - * The proxy does not need to send a request right now.
 - * (Threat 3:) The proxy withholds a request.
- o The server receives too many requests.
Reasons include:
 - * (Threat 4:) The proxy floods the server with requests.

A proxy eavesdropping or inferring information from messages it operates on has an impact on a server in the same way as on a client:

- o (Threat 5:) The proxy eavesdrops on the data in the response from the server.
- o (Threat 6:) The proxy measures the size, frequency or distribution of responses from the server.

2.1.2.1. Threat 1: Spoofing

With one exception (see below), this threat is REQUIRED to be mitigated by the security solution: the server MUST verify that the request is an "authentic request" before processing it.

The definition of an "authentic request" depends on the desired proxy functionality and protection level (Section 2.2), but usually means that the server can obtain proof for some or all of the following items:

- o that the proxy acts on behalf of a client;
- o that the data originates from the client and has not been altered on the way;
- o that the request has not been received previously.

The proof can, for example, involve a message authentication code that the proxy obtains from the client and includes in the request or a challenge-response roundtrip.

Exception: A CoAP proxy may make certain requests without acting on behalf of a client (e.g., to validate cached data). Since such a request does not originate from a client, the server cannot tell if the proxy sends the request according to specification or if it spoofs the request. It is up to the security solution how this issue is addressed.

2.1.2.2. Threat 2: Delaying

This threat is REQUIRED to be mitigated by the security solution; see Section 2.1.1.2.

2.1.2.3. Threat 3: Withholding

This threat is NOT REQUIRED to be mitigated by the security solution, since a server cannot tell if the proxy does not send a request because it has no work to do or if it intentionally withholds a request.

2.1.2.4. Threat 4: Flooding

This threat is NOT REQUIRED to be mitigated by the security solution in particular, but a server SHOULD generally defend against flooding attacks.

2.1.2.5. Threat 5: Eavesdropping

This threat is REQUIRED to be mitigated by the security solution; see Section 2.1.1.5.

2.1.2.6. Threat 6: Traffic Analysis

This threat is NOT REQUIRED to be mitigated by the security solution; see Section 2.1.1.6.

2.2. Solutions

A security solution has to find a trade-off between desired proxy functionality (such as caching) and the provided level of protection. From this trade-off results the definition of what constitutes an "authentic request" or "authentic response" and when a request or response is considered confidentiality protected.

This section presents two exemplary choices of trade-offs:

- o The first case focuses on a high protection level by tying requests and responses uniquely together and confidentiality protecting as much as possible, at the cost of reduced proxy functionality.
- o The second case aims to preserve proxy functionality as much as possible, at the cost of reduced confidentiality protection.

For both cases, this section presents an overview of the functionality and processing rules of the proxy and analyses the required authenticity and confidentiality properties of requests and responses. Due to space constraints, the analysis is limited to the CoAP header, the request and response options shown in Table 1, and the payload.

Requests	Responses
Accept	Content-Format
Content-Format	ETag
ETag	Location-Path
If-Match	Location-Query
If-None-Match	Max-Age
Observe	Observe
Proxy-Scheme	
Proxy-Uri	
Uri-Host	
Uri-Port	
Uri-Path	
Uri-Query	

Table 1: Analysed CoAP Options

Note that, since CoAP was not designed with end-to-end security in mind, a security solution extends the applicability of CoAP beyond its original scope.

2.2.1. Forwarding

In this case we study the functionality of a CoAP forward proxy and assume that caching is disabled. This is applicable to many security critical use cases where a response needs to be securely linked to a unique request from a client and cannot be re-used with another request.

There may be a unique response for each request (see Figure 6) or multiple responses for each request (see Figure 7).

2.2.1.1. Examples

Examples of the need for unique response for each request include alarm status retrieval and actuator command confirmation.

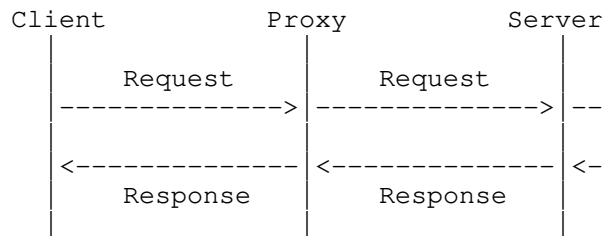


Figure 6: Message Flow with a Unique Response for Each Request

Example: Alarm status retrieval

Figure 6 can be seen as an illustration of a message exchange for a client requesting the alarm status (e.g., GET /alarm_status) from a server. Since the client wants to ensure that the alarm status received is reflecting the current alarm status and not a cached or spoofed response to the same resource, it must be able to verify that the received response is a response to this particular request made by the client. Therefore, the response must be securely linked to the request.

Example: Actuation confirmation

Another example for which Figure 6 serves as illustration is the confirmation of an actuator request. In this case a client, say in an industrial control system, requests a server that a valve should be turned to a certain level, e.g. PUT /valve_42/level with payload "3". In order for the client to correctly evaluate the result of a potential changed valve level, it is important that the client gets a confirmation how the server responded to the requested change, e.g., whether the request was performed or

not. Again, the client wants to ensure that the response is reflecting the result of this particular actuation request made by the client and not a cached or spoofed response. Therefore, the response must be securely linked to the request.

An example of the use of multiple responses for each request is in security critical monitoring scenarios where time synchronization cannot be guaranteed. By avoiding repeated requests from the same client to the same resource, constrained node resources and bandwidth is saved.

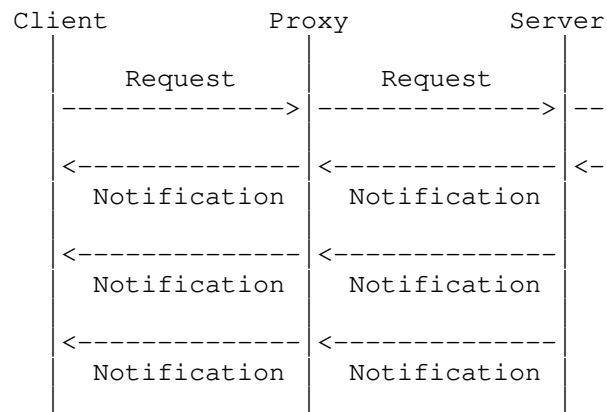


Figure 7: Message Flow of Notifications of Linked to a Unique Request

Example: Secure parameter monitoring

Figure 7 can be seen as an illustration of a message exchange for a client monitoring an important parameter measured by the server, e.g., in a medical or process industry application. The client makes a subscription request for a resource and the server responds with notifications, e.g. providing updates to the parameter on regular time intervals.

The client wants to ensure that the first received notification reflects the current parameter value and that subsequent notifications are timely updates of the initial request. Since notifications may be lost or reordered, the client needs to be able to verify the order of the messages, as sent by the server. By monitoring the received messages and the time they are received, the client can detect missing notifications and take appropriate action.

2.2.1.2. Functional Requirements

- FR1.1 The caching functionality MUST be inhibited; the CoAP option Max-Age of the responses SHALL be 0 (see Section 5.7.1 of [RFC7252]).
- FR1.2 To limit information leaking about the resource (see Section 2.2.1.5) the Proxy-Uri does not contain Uri-Path or Uri-Query.

2.2.1.3. Processing Rules

In this case, the desired proxy functionality is to forward a translated request to the determined destination. There are two modes of operation for requests: Either using the Proxy-Uri option (PR1.1) or using the Proxy-Scheme option together with the Uri-Host, Uri-Port, Uri-Path and Uri-Query options (PR1.2).

- PR1.1 The Proxy-Uri option contains the request URI including request scheme (e.g. "coaps://"); the Proxy-Scheme and Uri-* options are not present.

If the proxy is configured to forward requests to another proxy, then it keeps the Proxy-Uri option; otherwise, it splits the option into its components, adds the corresponding Uri-* options and removes the Proxy-Uri option. Then it makes the request using the request scheme indicated in the Proxy-Uri.

- PR1.2 The Proxy-Scheme option and the Uri-* options together contain the request URI; the Proxy-Uri option is not present.

If the proxy is configured to forward requests to another forwarding proxy, then it keeps the Proxy-Scheme and Uri-* options; otherwise, it removes the Proxy-Scheme option. Then it makes the request using the request scheme indicated in the removed Proxy-Scheme option.

- PR1.3 Responses are forwarded by the proxy, without any modification.

2.2.1.4. Authenticity

A request is considered authentic by the server (Section 2.1.2.1) if the server can obtain proof for all of the following items:

- A1.1 that the proxy acts on behalf of a client;

A1.2 that the following parts of the request originate from the client and have not been altered on the way:

- * the CoAP version,
- * the request method,
- * all options except Proxy-Uri, Proxy-Scheme, Uri-Host, Uri-Port, Uri-Path and Uri-Query, and
- * the payload, if any.

A1.3 that the effective request URI originates from the client and has not been altered on the way;

A1.4 that the request has not been received previously;

A1.5 that the request from the client to the proxy was sent recently.

A response is considered authentic by the client (Section 2.1.1.1) if the client can obtain proof for all of the following items:

A1.6 that the following parts of the response originate from the server and have not been altered on the way:

- * the CoAP version,
- * the response code,
- * all options, and
- * the payload, if any.

A1.7 that the response corresponds uniquely to the request sent by the client.

A1.8 that the response has not been received previously;

A1.9 that the response from the server to the proxy was sent recently;

A1.10 that the response is in sequence if there are multiple responses.

2.2.1.5. Confidentiality

The following parts of the message are confidentiality protected (Section 2.1.1.5):

- o all options except Proxy-Uri, Proxy-Scheme, Uri-Host and Uri-Port;
- o the payload, if any.

2.2.2. Caching

In this case we study caching: how a proxy may serve the same cached response to multiple clients requesting the same resource.

The caching functionality protects communication-constrained servers from repeated requests for the same resources, possibly originating from different clients. This saves system resources, bandwidth, and round-trip time.

There may be one response for each request (see Figure 8) or multiple responses for each request (see Figure 9).

2.2.2.1. Examples

The first example is a simple case of caching.

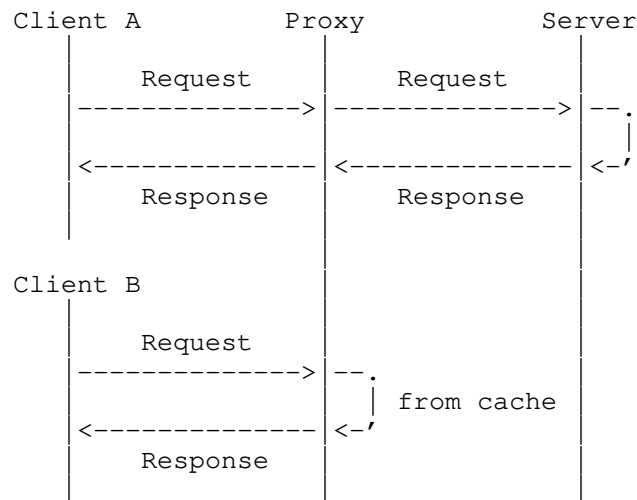


Figure 8: Message Flow for Cached Responses

Example: Caching

In Figure 8, client A requests the proxy to make a certain request to the server and to return the server's response. The proxy services the request by making a request message to the server according to the processing rules. If the server returns a cacheable response, then the proxy stores the response in its cache, performs any necessary translations, and forwards it to the client. Later, client B makes an equivalent request to the proxy that the proxy services by returning the response from its cache. Both client A and B want to verify that the response is valid.

In addition to multiple clients' requests being served by one response, each request may result in multiple responses. The difference compared to Section 2.2.1 is that in this example multiple clients may be served with the same response, further saving server resources.

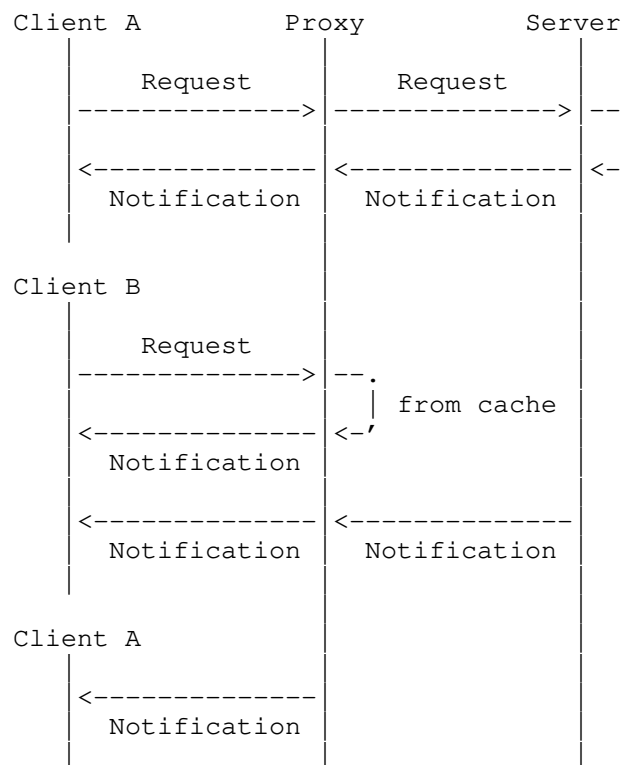


Figure 9: Message Flow for Observe with Multiple Observers

Example: Observe with caching

In Figure 9, the server exposes an observable resource (e.g., the current reading of a temperature sensor). Multiple clients are interested in the current state of the resource and observe it using the CoAP resource observation mechanism [RFC7641]. The goal is to keep the state observed by the clients closely in sync with the actual state of the resource at the server. Another goal is to minimize the burden on the server by moving the task to fan out notifications to multiple clients from the server to the proxy.

2.2.2.2. Functional Requirements

The security solution SHOULD protect requests and responses in a way that a proxy can perform the following tasks:

- FR2.1 Storing a cacheable response in a cache. This requires that the proxy is able to calculate the cache-key of the request. Cacheable responses include 2.05 (Content) responses and all error responses.
- FR2.2 Returning a fresh response from its cache without contacting the server.
- FR2.3 Performing validation of a response cached by the proxy as well as validation of a response cached by the client.
- FR2.4 Observing a resource on behalf of one or more clients.

2.2.2.3. Processing Rules

The proxy complies with the forwarding rules PR1.1 - 1.3 (Section 2.2.1.3) and the rules below. The rules below have priority.

- PR2.1 If the proxy receives a request where the cache key matches that of a cached fresh response, then the proxy with that response; otherwise, it makes a request towards the server.
- PR2.2 The proxy caches and forwards cacheable responses. If there is already a response in the cache with the cache key of the corresponding request, then the old response in the cache is marked as stale.
- PR2.3 If the proxy receives a request that contains an ETag option and the proxy has a fresh response with the same cache key and ETag, then the proxy replies to the request with a 2.03 (Valid) response without payload, else it forwards a translated request.

PR2.4 The proxy updates the Max-Age option according to the Max-Age associated with the resource representation it receives, decreasing its value to reflect the time spent in the cache.

PR2.5 If the request contains an Accept option and if there is a fresh response that matches the cache key for the corresponding request except for the Accept option and if the Content-Format of the response matches that of the Accept option, then the proxy forwards the cached response to the requesting client.

2.2.2.4. Authenticity

A request is considered authentic by the server (Section 2.1.2.1) if the server can obtain proof for all of the following items:

A2.1 that the following parts of the request originate from the client and have not been altered on the way:

- * the CoAP version,
- * the request method,
- * all options except ETag, Observe, Proxy-Uri, Proxy-Scheme, Uri-Host, Uri-Port, Uri-Path and Uri-Query, and
- * the payload, if any.

A2.2 that the effective request URI originates from the client and has not been altered on the way;

A response is considered authentic by the client (Section 2.1.1.1) if the client can obtain proof for all of the following items:

A2.3 that the following parts of the response originate from the server and have not been altered on the way:

- * the CoAP version,
- * the response code,
- * all options except Max-Age and Observe, and
- * the payload, if any.

A2.4 that the response matches the specifications of the request;

A2.5 that the data is fresh (when the response is cacheable);

A2.6 that the response is in sequence (when observing a resource).

2.2.2.5. Confidentiality

No parts of a request are confidentiality protected
(Section 2.1.2.5).

A response is considered confidentiality protected (Section 2.1.2.5)
if the payload of the response is confidentiality protected.

3. Publish-Subscribe

Much of the concerns about proxies as described previously in this document also applies to other kinds of intermediary nodes. In this section we study brokers in a publish-subscribe setting [I-D.ietf-core-coap-pubsub]. The case of combining brokers and proxies is out of scope for this version of the document.

There are different ways for a pub-sub broker to operate. We consider the following broker operations:

- o The broker receives a request for a topic from a subscriber.
- o The broker receives a request for a publication to a topic from a publisher and forwards the publication to the subscribers of the topic.

We consider the setting where there is a security association between publisher and subscriber such that the publications can be protected during transfer, see Figure 10.

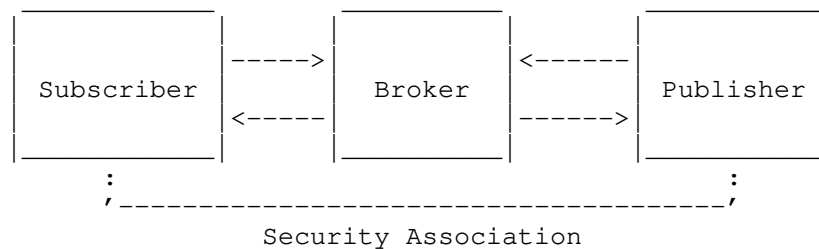


Figure 10: Publisher-to-Subscriber Security

Since there is no security association with the broker, we only consider the subscribe and publish functionality of the broker. Note that the broker needs to read the topic to accomplish this task.

3.1. Threats and Security Requirements

3.1.1. Subscriber-side

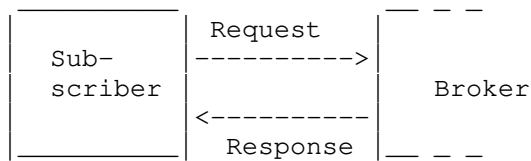


Figure 11: The Subscriber End

The subscriber sends a subscription request to the broker and waits for a response.

From the perspective of the subscriber, there are three possible flows:

- o The subscriber receives a response.
Reasons include:
 - * The broker duly processed the request and returns a response based on data it obtained from a publisher.
 - * The subscriber made a bad request and the broker returns an error response accordingly (e.g., 4.04 Not Found).
 - * The broker encountered an unexpected condition and returns an error response accordingly (e.g., 5.03 Service Unavailable).
 - * (Threat 1:) The broker spoofs a response.
 - * (Threat 2:) The broker duly processed the request but delays the return of a response.
- o The subscriber does not receive a response.
Reasons include:
 - * The subscriber times out too early.
 - * (Threat 3:) The broker withholds a response.
- o The subscriber receives too many responses.
Reasons include:
 - * (Threat 4:) The broker floods the subscriber with responses.

Furthermore, there are threats related to privacy:

- o (Threat 5:) The broker eavesdrops on the data in the request from the subscriber.

- o (Threat 6:) The broker measures the size, frequency or distribution of requests from the subscriber.

Note that "topic poisoning" -- the case of storing injected incorrect publications -- is covered from the point of view of the subscriber: it may result in the subscriber receiving a spoofed message, or being flooded, or affect other nodes such that the subscriber times out too early.

3.1.1.1. Threat 1: Spoofing

With one exception (see below), this threat is REQUIRED to be mitigated by the security solution: the subscriber MUST verify that a response is an "authentic publication" before processing it.

The definition of an "authentic publication" depends on the setting (Section 3.2), but usually means that the subscriber can obtain proof for some or all of the following items:

- o that the data matches the specifications of the request (such as the topic);
- o that the data originates from a publisher that is authorized to publish to the topic;
- o that the data has not been altered on the way between publisher and subscriber;
- o that the data is fresh (when the data is cacheable);
- o that the data is in sequence (when observing a topic).

The proof can, for example, include a message authentication code that the proxy obtains from the origin server and includes in the response or an additional challenge-response roundtrip.

Exception: A CoAP server like the broker is specified to return an error response (such as 4.04 Not Found or 5.03 Service Unavailable) when it encounters an error condition. Since the condition occurs at the broker and not at the publisher, the response will not be an "authentic response" according to the above definition. Thus, a subscriber cannot tell if the broker sends the error response according to specification or if it spoofs the response. This threat is NOT REQUIRED to be mitigated by the security solution.

3.1.1.2. Threat 2: Delaying

This threat is NOT REQUIRED to be mitigated by the security solution.

3.1.1.3. Threat 3: Withholding

This threat is NOT REQUIRED to be mitigated by the security solution, since a subscriber cannot tell if the broker does not send a response because it is hasn't received a publication from the publisher yet or if it intentionally withholds the response.

3.1.1.4. Threat 4: Flooding

A CoAP client like the subscriber is specified to reject any response that it does not expect. This can happen before the subscriber verifies if the response is authentic. Therefore, a flood of responses is primarily a threat to the system resources of the client, in particular to its energy. This threat is NOT REQUIRED to be mitigated by the security solution, but a subscriber SHOULD generally defend against flooding attacks.

3.1.1.5. Threat 5: Eavesdropping

This threat is NOT REQUIRED to be mitigated: The broker needs to read all parts of the request from the subscriber to accomplish its task.

It is RECOMMENDED that applications analyse the risks associated with application information leaking from the messages flow and assess the feasibility to protect against various threats, e.g., by obfuscating topic content.

3.1.1.6. Threat 6: Traffic Analysis

This threat is NOT REQUIRED to be mitigated by the security solution.

It is RECOMMENDED that applications analyse the risks associated with application information leaking from the messages flow and assess the feasibility to protect against various threats, e.g., by obfuscating parameters transported in plain text, aligning message flow and traffic between the different cases, adding padding so different messages become indistinguishable, etc.

3.1.2. Publisher-side

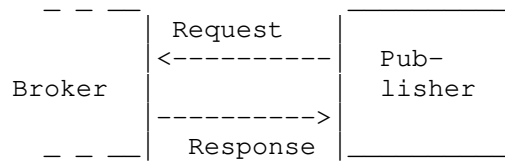


Figure 12: The Publisher End

The publisher sends a publication request to the broker and waits for a response.

The threat of the broker eavesdropping on the data in the publication request is REQUIRED to be mitigated by the security solution: publishers MUST confidentiality protect the data in the requests they send. This excludes parts that the broker needs to read to perform its job, e.g., the topic.

The threat of the broker measuring the size, frequency or distribution of publication requests is NOT REQUIRED to be mitigated by the security solution; see Section 3.1.1.6.

The broker is in full control of the response and may therefore arbitrarily spoof, delay, or withhold it. This threat is NOT REQUIRED to be mitigated. For example, a proof that the broker has notified all subscribers is NOT REQUIRED.

3.2. Solutions

3.2.1. Brokering

In this case we study brokering: how a broker may serve the same publication to multiple subscribers observing the same topic.

The brokering functionality protects communication-constrained publishers from repeated requests for the same resources, possibly originating from different subscribers. This saves system resources, bandwidth, and round-trip time.

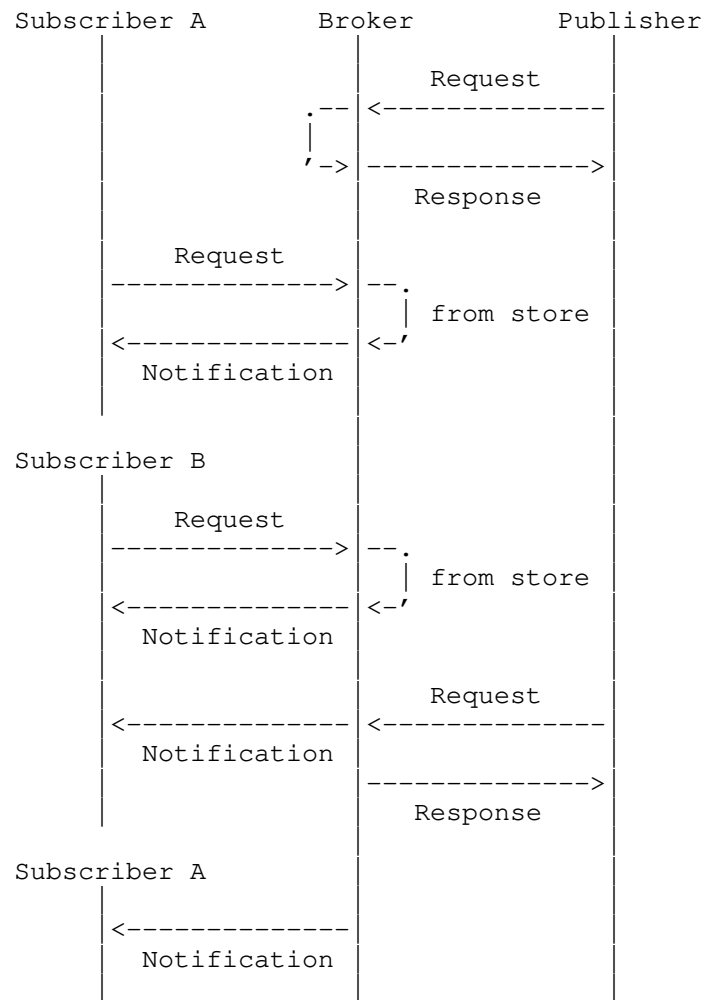


Figure 13: Message Flow for Publish Subscribe

Example

In Figure 13, the publisher publishes to a topic (e.g., the current reading of a temperature sensor). Multiple subscribers are interested in the current state of the topic and observe the topic as specified in [I-D.ietf-core-coap-pubsub]. The goal is to keep the state observed by the subscribers closely in sync with the actual state of the resource at the publisher. Another goal is to minimize the burden on the publisher by moving the task to fan out notifications to multiple subscribers from the publisher to the broker.

3.2.1.1. Functional Requirements

The security solution SHOULD protect subscription and publication requests in a way that a broker can perform the following tasks:

FR3.1 Storing publications. This requires that the broker is able to read the topic of the request.

FR3.2 Returning a stored publication without contacting the publisher.

3.2.1.2. Processing Rules

The broker complies with the following rules:

PR3.1 If the broker receives a request where the topic matches that of a cached publication, then the broker responds with that publication.

PR3.2 The broker caches and forwards publication notifications.

3.2.1.3. Authenticity

A publication is considered authentic by the subscriber if the subscriber can obtain proof for all all of the following items:

A3.1 that the payload is associated to the topic;

A3.2 that the payload has not been altered since published;

A3.3 that the publication is in sequence.

3.2.1.4. Confidentiality

The payload of a publication request is confidentiality protected.

4. Security Considerations

This document is about security; as such, there are no additional security considerations.

5. IANA Considerations

This document includes no request to IANA.

6. References

6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc7641>>.

6.2. 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-02 (work in progress), July 2017.
- [I-D.mattsson-core-coap-actuators] Mattsson, J., Fornehed, J., Selander, G., and F. Palombini, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-02 (work in progress), November 2016.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<http://www.rfc-editor.org/info/rfc7228>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<http://www.rfc-editor.org/info/rfc8152>>.

Acknowledgments

Thanks to Ari Keranen, John Mattsson, Jim Schaad and Ludwig Seitz for helpful comments and discussions that have shaped the document.

Authors' Addresses

Goeran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen 28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

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

P. van der Stok
consultant
K. Hartke
Universitaet Bremen TZI
February 26, 2018

"Pending" Responses for the Constrained Application Protocol (CoAP)
draft-hartke-core-pending-02

Abstract

This document proposes a new type of response for the Constrained Application Protocol (CoAP) called a "Pending" response. A CoAP server can use a Pending response to indicate that it has accepted a request but has not yet started processing it or that processing the request will take longer than a client is typically willing to wait for a response.

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. Terminology	3
2. Pending Responses	3
2.1. Observing Resources	4
3. Security Considerations	4
4. IANA Considerations	5
5. References	5
5.1. Normative References	5
5.2. Informative References	5
Authors' Addresses	6

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a request/response protocol not unlike HTTP. CoAP defines no upper bound for the time between a request and the resulting response. For example, a CoAP-over-UDP server is expected to return an empty Acknowledgement to the client if it cannot provide a response right away, but there is no limit on the time when the server should return the Separate Response.

In particular in the case of requests with long processing times, a CoAP client faces the problem that it cannot easily determine how long it should wait for the response and whether the CoAP server is actually still processing the request. Long processing times occur, for example, when requests need manual intervention to authorize their processing, or when they perform a long sequence of remote actions. An example for this is the "possibly long" authorization request specified in EST-coaps [I-D.vanderstok-ace-coap-est].

This document proposes a new kind of response in CoAP, called a "Pending" response. The semantics of this response are modelled after the HTTP 202 (Accepted) status code [RFC7231]:

The 202 (Accepted) status code indicates that the request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place. [...] The representation sent with this response ought to describe the request's current status and point to (or embed) a status monitor that can provide the user with an estimate of when the request will be fulfilled.

Pending responses are not intended for overload cases, which are better handled by the 5.03 (Service Unavailable) response code.

1.1. Terminology

Readers are expected to be familiar with the terms and concepts described in [RFC7252] and [RFC7641].

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].

2. Pending Responses

A Pending response is denoted by a response code in the 2.xx range and a Content-Format Option that is set to content-format ID TBD1.

A 2.01 (Creation Pending) response in reply to a POST request indicates that the result of processing the request is not available yet, for example, because the server needs more time to process the request than a client is typically willing to wait for a response. The server MAY specify a location using the Location-* options where the result will become available. If the server does not specify a location, the result will become available at the target resource of the POST request. To retrieve the result, the client MAY poll or observe the resource at this location using the GET request method.

A 2.02 (Deletion Pending) response in reply to a DELETE request indicates that the server has accepted the request but the target has not been fully deleted yet.

A 2.04 (Change Pending) response in reply to a POST or PUT request indicates that the server has accepted the request but the result of processing the request is not available yet.

A 2.05 (Content Pending) response in reply to GET request indicates that the target resource exists but a representation of the resource is not available yet. The Max-Age Option indicates after what time a client should retry its GET request to retrieve the representation. The client MAY observe the resource [RFC7641] to get notified when the representation becomes available (see Section 2.1 for details).

The payload of a Pending response MAY be a brief human-readable diagnostic message, explaining the situation, or MUST be absent.

The cacheability of Pending responses is as specified for the respective response code.

2.1. 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 Pending) 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.

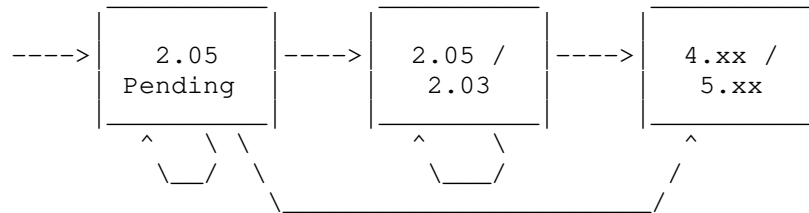


Figure 1: Sequence of Notifications

Unless the server is unwilling to add the client to the list of observers, each 2.05 (Content Pending) notification MUST include an Observe Option with a sequence number as specified in [RFC7641]. Otherwise, the registration request falls back to a normal GET request.

3. Security Considerations

This section analyses the possible threats related to Pending responses. It is meant to inform protocol and application developers about the security limitations of the response code as described in this document.

A Pending response is subject to the same general security considerations as all CoAP responses as described in Section 11 of [RFC7252]. Specifically, the security considerations for the response code are closest to those of the Observe Option as stated in Section 7 of [RFC7641], because the server stores additional state over an extended period.

Pending responses are secured following the recommendations for the existing CoAP response codes as specified in Section 9 of [RFC7252]. When additional security techniques are standardized for CoAP (e.g., based on object security), these are then also available for securing the responses.

4. IANA Considerations

This document adds the content-format used to signal Pending responses to the "CoAP Content-Formats" registry.

Media Type	Content Coding	ID	Reference
-	-	TBD1	[This Document]

New CoAP Content-Formats

TBD1 is taken from the "First Come First Served" range of the "CoAP Content-Formats" registry.

5. References

5.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [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>>.

5.2. Informative References

- [I-D.vanderstok-ace-coap-est] Stok, P., Kampanakis, P., Kumar, S., Richardson, M., Furuheid, M., and S. Raza, "EST over secure CoAP (EST-coaps)", draft-vanderstok-ace-coap-est-04 (work in progress), January 2018.
- [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>>.

Authors' Addresses

Peter van der Stok
consultant

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

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

CoRE Working Group
Internet-Draft
Intended status: Standards Track
Expires: 5 November 2022

M. Koster
SmartThings
A. Keranen
J. Jimenez
Ericsson
4 May 2022

Publish-Subscribe Broker for the Constrained Application Protocol (CoAP)
draft-ietf-core-coap-pubsub-10

Abstract

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

Status of This Memo

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

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

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

This Internet-Draft will expire on 5 November 2022.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Notational Conventions	3
3. Terminology	3
4. Architecture	4
4.1. CoAP Pub/sub Architecture	4
4.2. CoAP Pub/sub Broker	5
4.3. CoAP Pub/sub Client	5
4.4. CoAP Pub/sub Topic	5
4.5. Brokerless Pub/sub	6
5. CoAP Pub/sub REST API	7
5.1. DISCOVERY	7
5.2. CREATE	9
5.3. PUBLISH	11
5.4. SUBSCRIBE	14
5.5. UNSUBSCRIBE	16
5.6. READ	17
5.7. REMOVE	18
6. CoAP Pub/sub Operation with Resource Directory	20
7. Sleep-Wake Operation	20
8. Simple Flow Control	21
9. Security Considerations	21
10. IANA Considerations	22
10.1. Resource Type value 'core.ps'	22
10.2. Resource Type value 'core.ps.discover'	22
11. Acknowledgements	23
12. References	23
12.1. Normative References	23
12.2. Informative References	24
Authors' Addresses	24

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] supports machine-to-machine communication across networks of constrained devices. CoAP uses a request/response model where clients make requests to servers in order to request actions on resources. Depending on the situation the same device may act either as a server, a client, or both.

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

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

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

2. Notational Conventions

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

3. Terminology

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 [RFC9176]. The URI template format [RFC6570] is used to describe the REST API defined in this specification.

This specification makes use of the following additional terminology:

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

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

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

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

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

4. Architecture

4.1. 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.

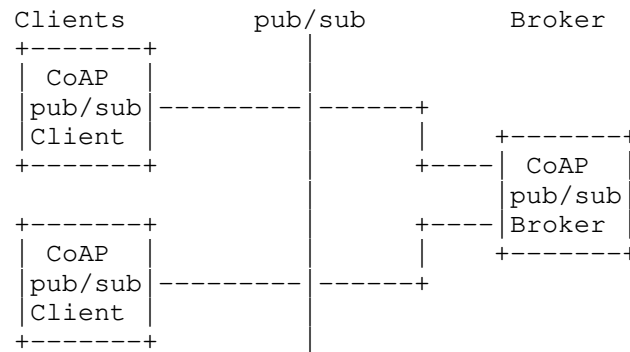


Figure 1: CoAP pub/sub Architecture

4.2. 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.

4.3. 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.

4.4. CoAP Pub/sub Topic

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

Every CoAP pub/sub topic has an associated link, consisting of a reference path on the Broker using URI path [RFC3986] construction and link attributes [RFC6690]. Every topic is associated with zero or more stored representations and a content-format specified in the

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

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

4.5. Brokerless Pub/sub

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

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

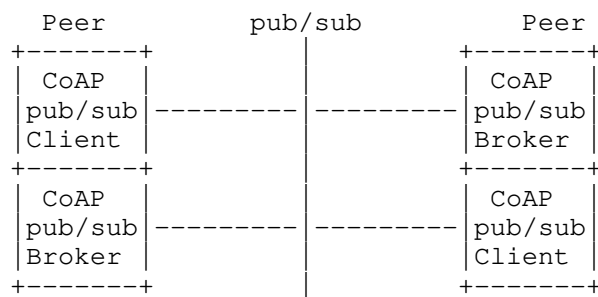


Figure 2: Brokerless pub/sub

5. 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.

5.1. DISCOVERY

CoAP pub/sub Clients discover CoAP pub/sub Brokers by using CoAP Simple Discovery or through a Resource Directory (RD) [RFC9176]. A CoAP pub/sub Broker SHOULD indicate its presence and availability on a network by exposing a link to the entry point of its pub/sub API at its .well-known/core location [RFC6690]. A CoAP pub/sub Broker MAY register its pub/sub REST API entry point with a Resource Directory. Figure 3 shows an example of a client discovering a local pub/sub API using CoAP Simple Discovery. A Broker wishing to advertise the CoAP pub/sub API for Simple Discovery or through a Resource Directory MUST use the link relation `rt=core.ps`. A Broker MAY advertise its supported content formats and other attributes in the link to its pub/sub API.

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

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

Topics in the broker may be created in hierarchies (see Section 5.2) with parent topics having sub-topics. For a discovery the broker may choose to not expose the sub-topics in order to limit amount of topic links sent in a discovery response. The client can then perform discovery for the parent topics it wants to discover the sub-topics.

The DISCOVER interface is specified as follows:

Interaction: Client -> Broker
 Method: GET
 URI Template: {+ps}/{+topic}{?q*}
 URI Template Variables:
 ps := Pub/sub REST API entry point (optional).
 Entry point of the pub/sub topic discovery API.
 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

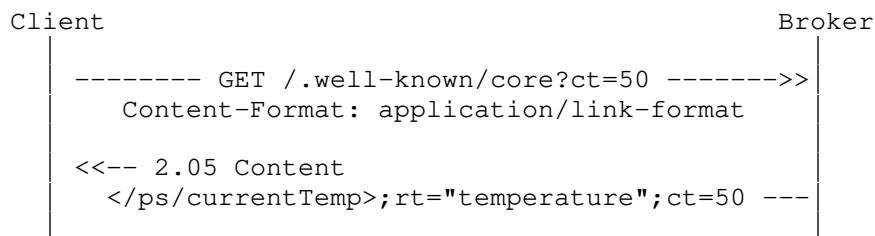


Figure 5: Example of DISCOVER topic

5.2. CREATE

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

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

Topic hierarchies can be created by creating parent topics. A parent topic is created with a POST using ct (Content Format) link attribute value which describes a supported serialization of the CoRE link format [RFC6690], such as application/link-format (ct=40) or its JSON or CBOR serializations. If a topic is created which describes a link serialization, that topic may then have sub-topics created under it as shown in Figure 7.

Only one level in the topic hierarchy may be created as a result of a CREATE operation, unless create on PUBLISH is supported (see Section 5.3). The topic string used in the link target MUST NOT contain the "/" character.

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

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

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

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

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

The CREATE interface is specified as follows:

Interaction: Client -> Broker

Method: POST

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

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

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

Content-Format: application/link-format

Payload: The desired topic to CREATE

The following response codes are defined for the CREATE operation:

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

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

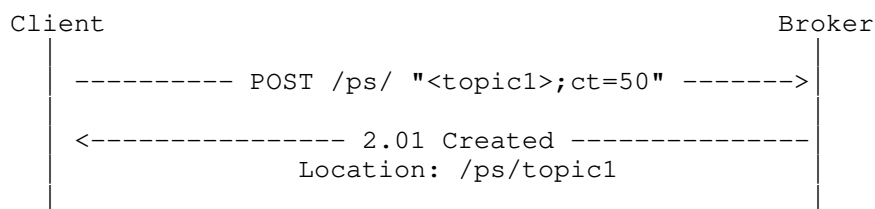


Figure 6: Example of CREATE topic

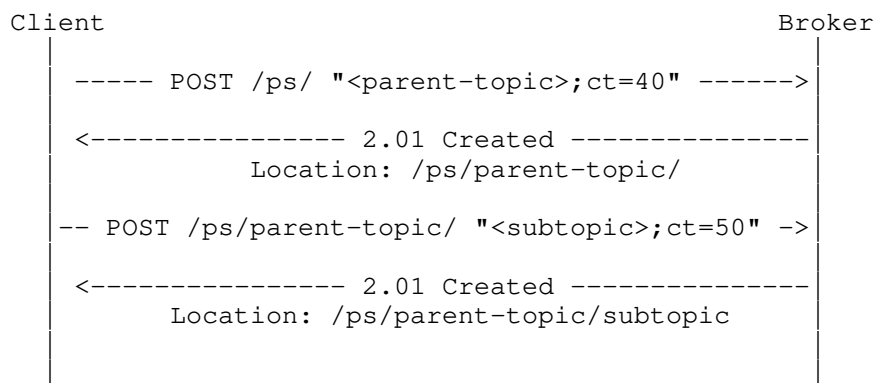


Figure 7: Example of CREATE of topic hierarchy

5.3. 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 8 is implemented.

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

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

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

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

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

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

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

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

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

The PUBLISH operation is specified as follows:

Interaction: Client -> Broker

Method: PUT, POST

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

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

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

Content-Format: Any valid CoAP content format

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

The following response codes are defined for the PUBLISH operation:

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

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

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

Failure: 4.29 "Too Many Requests". The client should slow down the rate of publish messages for this topic (see Section 8).

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.

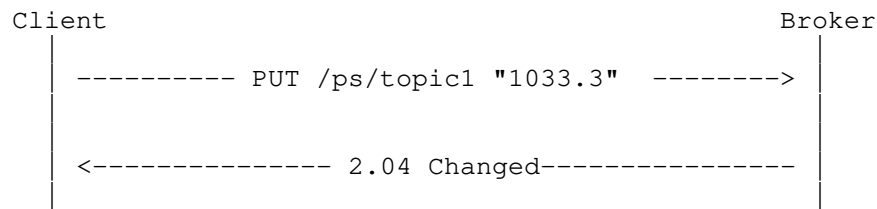


Figure 8: Example of PUBLISH

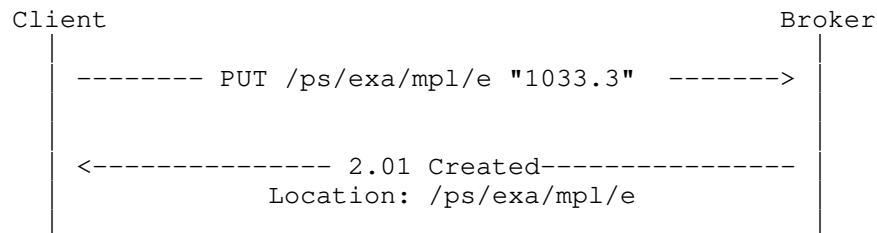


Figure 9: Example of CREATE on PUBLISH

5.4. SUBSCRIBE

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

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

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

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

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

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

The SUBSCRIBE operation is specified as follows:

Interaction: Client -> Broker

Method: GET

Options: Observe:0

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

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

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

The following response codes are defined for the SUBSCRIBE operation:

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

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

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

Figure 10 shows an example of Client2 subscribing to "topic1" and receiving a response from the Broker, with a subsequent notification. The subscribe response from the Broker uses the last stored value associated with the topic1. The notification from the Broker is sent in response to the publish received from Client1.

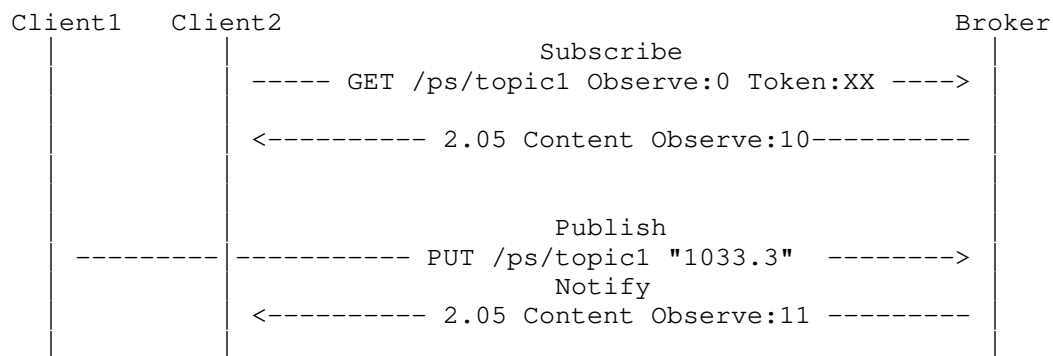


Figure 10: Example of SUBSCRIBE

5.5. 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.

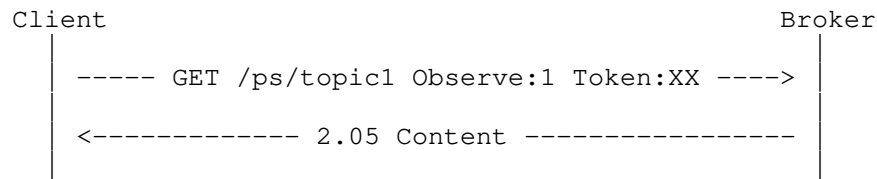


Figure 11: Example of UNSUBSCRIBE

5.6. READ

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

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

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

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

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

The READ operation is specified as follows:

Interaction: Client -> Broker

Method: GET

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

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

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

The following response codes are defined for the READ operation:

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

Failure: 4.00 "Bad Request". Malformed request.

Failure: 4.01 "Unauthorized". Authorization failure.

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

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

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

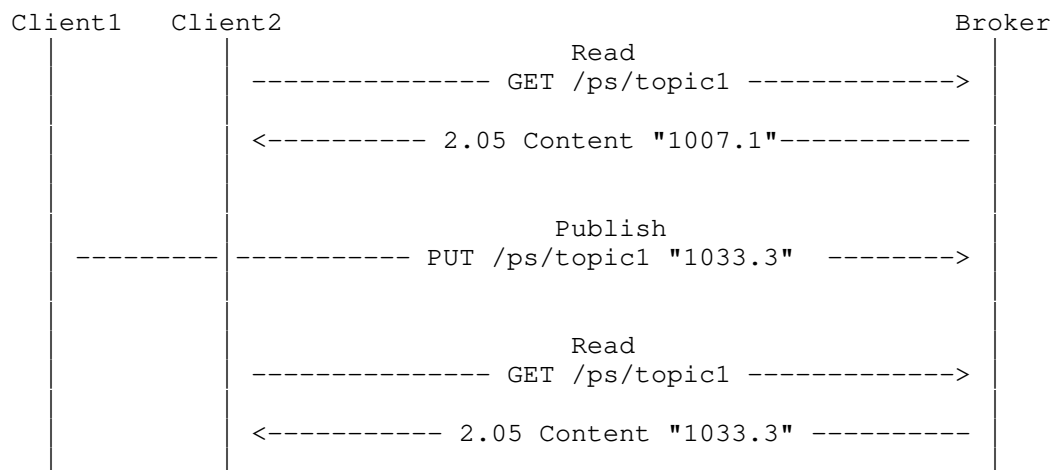


Figure 12: Example of READ

5.7. 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.

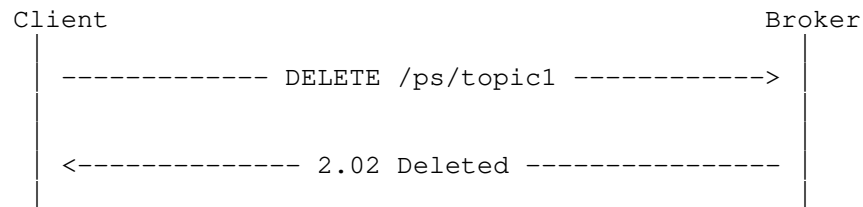


Figure 13: Example of REMOVE

6. 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) [RFC9176] 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 5.1 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 [RFC9176] for further details.

7. 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.

8. Simple Flow Control

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

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

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

9. Security Considerations

CoAP pub/sub re-uses CoAP [RFC7252], CoRE Resource Directory [RFC9176], 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.ietf-ace-pubsub-profile]. An example application that uses the CoAP pub/sub Broker and relies on end-to-end object security is described in [RFC8387]. When only end-to-end encryption is necessary and the CoAP Broker is trusted, Payload Only Protection (Mode:PAYL) could be used. The Publisher would wrap only the payload before sending it to the Broker and set the option Content-Format to application/smpayl. Upon receipt, the Broker can read the unencrypted CoAP header to forward it to the subscribers.

10. 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.

10.1. Resource Type value 'core.ps'

- * Attribute Value: core.ps
- * Description: Section 5 of [[This document]]
- * Reference: [[This document]]
- * Notes: None

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

- * Attribute Value: core.ps.discover
- * Description: Section 5 of [[This document]]
- * Reference: [[This document]]
- * Notes: None

11. 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.

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>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8516] Keranen, A., "'Too Many Requests' Response Code for the Constrained Application Protocol", RFC 8516, DOI 10.17487/RFC8516, January 2019, <<https://www.rfc-editor.org/info/rfc8516>>.

- [RFC9176] Amsuess, C., Ed., Shelby, Z., Koster, M., Bormann, C., and P. van der Stok, "Constrained RESTful Environments (CoRE) Resource Directory", RFC 9176, DOI 10.17487/RFC9176, April 2022, <<https://www.rfc-editor.org/info/rfc9176>>.

12.2. Informative References

- [I-D.ietf-ace-pubsub-profile]
Palombini, F. and C. Sengul, "Pub-Sub Profile for Authentication and Authorization for Constrained Environments (ACE)", Work in Progress, Internet-Draft, draft-ietf-ace-pubsub-profile-04, 29 December 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-pubsub-profile-04.txt>>.
- [RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <<https://www.rfc-editor.org/info/rfc5988>>.
- [RFC8387] Sethi, M., Arkko, J., Keranen, A., and H. Back, "Practical Considerations and Implementation Experiences in Securing Smart Object Networks", RFC 8387, DOI 10.17487/RFC8387, May 2018, <<https://www.rfc-editor.org/info/rfc8387>>.

Authors' Addresses

Michael Koster
SmartThings
Email: Michael.Koster@smarththings.com

Ari Keranen
Ericsson
Email: ari.keranen@ericsson.com

Jaime Jimenez
Ericsson
Email: jaime.jimenez@ericsson.com

CORE
Internet-Draft
Updates: 7641, 7959 (if approved)
Intended status: Standards Track
Expires: June 21, 2018

C. Bormann
Universitaet Bremen TZI
S. Lemay
Zebra Technologies
H. Tschofenig
ARM Ltd.
K. Hartke
Universitaet Bremen TZI
B. Silverajan
Tampere University of Technology
B. Raymor, Ed.
December 18, 2017

CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets
draft-ietf-core-coap-tcp-tls-11

Abstract

The Constrained Application Protocol (CoAP), although inspired by HTTP, was designed to use UDP instead of TCP. The message layer of the CoAP over UDP protocol includes support for reliable delivery, simple congestion control, and flow control.

Some environments benefit from the availability of CoAP carried over reliable transports such as TCP or TLS. This document outlines the changes required to use CoAP over TCP, TLS, and WebSockets transports. It also formally updates RFC 7641 for use with these transports and RFC 7959 to enable the use of larger messages over a reliable transport.

Status of This Memo

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

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

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

This Internet-Draft will expire on June 21, 2018.

Copyright Notice

Copyright (c) 2017 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. Conventions and Terminology	6
3. CoAP over TCP	7
3.1. Messaging Model	7
3.2. Message Format	8
3.3. Message Transmission	10
3.4. Connection Health	11
4. CoAP over WebSockets	12
4.1. Opening Handshake	13
4.2. Message Format	14
4.3. Message Transmission	15
4.4. Connection Health	15
5. Signaling	15
5.1. Signaling Codes	16
5.2. Signaling Option Numbers	16
5.3. Capabilities and Settings Messages (CSM)	16
5.4. Ping and Pong Messages	19
5.5. Release Messages	20
5.6. Abort Messages	21
5.7. Signaling examples	22
6. Block-wise Transfer and Reliable Transports	23
6.1. Example: GET with BERT Blocks	24
6.2. Example: PUT with BERT Blocks	25
7. Observing Resources over Reliable Transports	25
7.1. Notifications and Reordering	26
7.2. Transmission and Acknowledgements	26
7.3. Freshness	26
7.4. Cancellation	27
8. CoAP over Reliable Transport URIs	27
8.1. coap+tcp URI scheme	28
8.2. coaps+tcp URI scheme	28

8.3.	coap+ws URI scheme	29
8.4.	coaps+ws URI scheme	30
8.5.	Uri-Host and Uri-Port Options	31
8.6.	Decomposing URIs into Options	31
8.7.	Composing URIs from Options	32
9.	Securing CoAP	32
9.1.	TLS binding for CoAP over TCP	33
9.2.	TLS usage for CoAP over WebSockets	34
10.	Security Considerations	34
10.1.	Signaling Messages	34
11.	IANA Considerations	34
11.1.	Signaling Codes	34
11.2.	CoAP Signaling Option Numbers Registry	35
11.3.	Service Name and Port Number Registration	36
11.4.	Secure Service Name and Port Number Registration	37
11.5.	URI Scheme Registration	38
11.6.	Well-Known URI Suffix Registration	40
11.7.	ALPN Protocol Identifier	40
11.8.	WebSocket Subprotocol Registration	40
11.9.	CoAP Option Numbers Registry	41
12.	References	41
12.1.	Normative References	41
12.2.	Informative References	43
	Appendix A. CoAP over WebSocket Examples	45
	Appendix B. Change Log	48
	B.1. Since draft-ietf-core-coap-tcp-tls-02	48
	B.2. Since draft-ietf-core-coap-tcp-tls-03	48
	B.3. Since draft-ietf-core-coap-tcp-tls-04	48
	B.4. Since draft-ietf-core-coap-tcp-tls-05	48
	B.5. Since draft-ietf-core-coap-tcp-tls-06	49
	B.6. Since draft-ietf-core-coap-tcp-tls-07	49
	Acknowledgements	49
	Contributors	49
	Authors' Addresses	50

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments, assuming that UDP [RFC0768] can be used unimpeded, as can the Datagram Transport Layer Security protocol (DTLS [RFC6347]) over UDP. The use of CoAP over UDP is focused on simplicity, has a low code footprint, and a small over-the-wire message size.

The primary reason for introducing CoAP over TCP [RFC0793] and TLS [RFC5246] is that some networks do not forward UDP packets. Complete blocking of UDP happens in between about 2% and 4% of terrestrial access networks, according to [EK2016]. UDP impairment is especially

concentrated in enterprise networks and networks in geographic regions with otherwise challenged connectivity. Some networks also rate-limit UDP traffic, as reported in [BK2015] and deployment investigations related to the standardization of QUIC revealed numbers around 0.3 % [SW2016].

The introduction of CoAP over TCP also leads to some additional effects that may be desirable in a specific deployment:

- o Where NATs are present along the communication path, CoAP over TCP leads to different NAT traversal behavior than CoAP over UDP. NATs often calculate expiration timers based on the transport layer protocol being used by application protocols. Many NATs maintain TCP-based NAT bindings for longer periods based on the assumption that a transport layer protocol, such as TCP, offers additional information about the session lifecycle. UDP, on the other hand, does not provide such information to a NAT and timeouts tend to be much shorter [HomeGateway]. According to [HomeGateway] the mean for TCP and UDP NAT binding timeouts is 386 minutes (TCP) and 160 seconds (UDP). Shorter timeout values require keepalive messages to be sent more frequently. Hence, the use of CoAP over TCP requires less frequent transmission of keep-alive messages.
- o TCP utilizes more sophisticated congestion and flow control mechanisms than the default mechanisms provided by CoAP over UDP, which is useful for the transfer of larger payloads. (Work is, however, ongoing to add advanced congestion control to CoAP over UDP as well, see [I-D.ietf-core-cocoa].)

Note that the use of CoAP over UDP (and CoAP over DTLS over UDP) is still the recommended transport for use in constrained node networks, particularly when used in concert with blockwise transfer. CoAP over TCP is applicable for those cases where the networking infrastructure leaves no other choice. The use of CoAP over TCP leads to a larger code size, more roundtrips, increased RAM requirements and larger packet sizes. Developers implementing CoAP over TCP are encouraged to consult [I-D.gomez-lwig-tcp-constrained-node-networks] for guidance on low-footprint TCP implementations for IoT devices.

Standards based on CoAP such as Lightweight Machine to Machine [LWM2M] currently use CoAP over UDP as a transport; adding support for CoAP over TCP enables them to address the issues above for specific deployments and to protect investments in existing CoAP implementations and deployments.

Although HTTP/2 could also potentially address the need for enterprise firewall traversal, there would be additional costs and

delays introduced by such a transition from CoAP to HTTP/2. Currently, there are also fewer HTTP/2 implementations available for constrained devices in comparison to CoAP. Since CoAP also support group communication using IP layer multicast and unreliable communication IoT devices would have to support HTTP/2 in addition to CoAP.

Furthermore, CoAP may be integrated into a Web environment where the front-end uses CoAP over UDP from IoT devices to a cloud infrastructure and then CoAP over TCP between the back-end services. A TCP-to-UDP gateway can be used at the cloud boundary to communicate with the UDP-based IoT device.

Finally, CoAP applications running inside a web browser may be without access to connectivity other than HTTP. In this case, the WebSocket protocol [RFC6455] may be used to transport CoAP requests and responses, as opposed to cross-proxying them via HTTP to an HTTP-to-CoAP cross-proxy. This preserves the functionality of CoAP without translation, in particular the Observe mechanism [RFC7641].

To address the above-mentioned deployment requirements, this document defines how to transport CoAP over TCP, CoAP over TLS, and CoAP over WebSockets. For these cases, the reliability offered by the transport protocol subsumes the reliability functions of the message layer used for CoAP over UDP. (Note that both for a reliable transport and the CoAP over UDP message layer, the reliability offered is per transport hop: where proxies -- see Sections 5.7 and 10 of [RFC7252] -- are involved, that layer's reliability function does not extend end-to-end.) Figure 1 illustrates the layering:

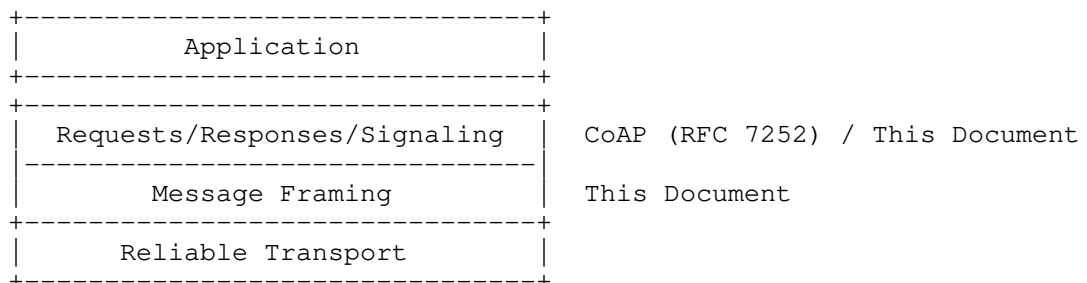


Figure 1: Layering of CoAP over Reliable Transports

This document specifies how to access resources using CoAP requests and responses over the TCP, TLS and WebSocket protocols. This allows connectivity-limited applications to obtain end-to-end CoAP connectivity either by communicating CoAP directly with a CoAP server accessible over a TCP, TLS or WebSocket connection or via a CoAP

intermediary that proxies CoAP requests and responses between different transports, such as between WebSockets and UDP.

Section 7 updates the "Observing Resources in the Constrained Application Protocol" [RFC7641] specification for use with CoAP over reliable transports. [RFC7641] is an extension to the CoAP protocol that enables CoAP clients to "observe" a resource on a CoAP server. (The CoAP client retrieves a representation of a resource and registers to be notified by the CoAP server when the representation is updated.)

2. Conventions and 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 assumes that readers are familiar with the terms and concepts that are used in [RFC6455], [RFC7252], [RFC7641], and [RFC7959].

The term "reliable transport" is used only to refer to transport protocols, such as TCP, which provide reliable and ordered delivery of a byte-stream.

Block-wise Extension for Reliable Transport (BERT):

BERT extends [RFC7959] to enable the use of larger messages over a reliable transport.

BERT Option:

A Block1 or Block2 option that includes an SZX value of 7.

BERT Block:

The payload of a CoAP message that is affected by a BERT Option in descriptive usage (see Section 2.1 of [RFC7959]).

Transport Connection:

Underlying reliable byte stream connection, as directly provided by TCP, or indirectly via TLS or WebSockets.

Connection:

Transport Connection, unless explicitly qualified otherwise.

Connection Initiator:

The peer that opens a Transport Connection, i.e., the TCP active opener, TLS client, or WebSocket client.

Connection Acceptor:

The peer that accepts the Transport Connection opened by the other peer, i.e., the TCP passive opener, TLS server, or WebSocket server.

3. CoAP over TCP

The request/response interaction model of CoAP over TCP is the same as CoAP over UDP. The primary differences are in the message layer. The message layer of CoAP over UDP supports optional reliability by defining four types of messages: Confirmable, Non-confirmable, Acknowledgement, and Reset. In addition, messages include a Message ID to relate Acknowledgments to Confirmable messages and to detect duplicate messages.

The management of the transport connections is left to the application, i.e., the present specification does not describe how an application decides to open a connection or to re-open another one in the presence of failures (or what it would deem to be a failure, see also Section 5.4). In particular, the Connection Initiator need not be the client of the first request placed on the connection. Some implementations will want to implement a dynamic connection management similar to the one described in Section 6 of [RFC7230] for HTTP, opening a connection when the first client request is ready to be sent and reusing that for further messages for a while, until no message is sent for a certain time and no requests are outstanding (possibly with a configurable idle time) and a release process is started (Section 5.5). In implementations of this kind, connection releases or aborts may not be indicated as errors to the application but may simply be handled by automatic reconnection once the need arises again. Other implementations may be based on configured connections that are kept open continuously and lead to management system notifications on release or abort. The protocol defined in the present specification is intended to work with either model (or other, application-specific connection management models).

3.1. Messaging Model

Conceptually, CoAP over TCP replaces most of the message layer of CoAP over UDP with a framing mechanism on top of the byte-stream provided by TCP/TLS, conveying the length information for each message that on datagram transports is provided by the UDP/DTLS datagram layer.

TCP ensures reliable message transmission, so the message layer of CoAP over TCP is not required to support acknowledgements or to detect duplicate messages. As a result, both the Type and Message ID

fields are no longer required and are removed from the CoAP over TCP message format.

Figure 2 illustrates the difference between CoAP over UDP and CoAP over reliable transport. The removed Type and Message ID fields are indicated by dashes.

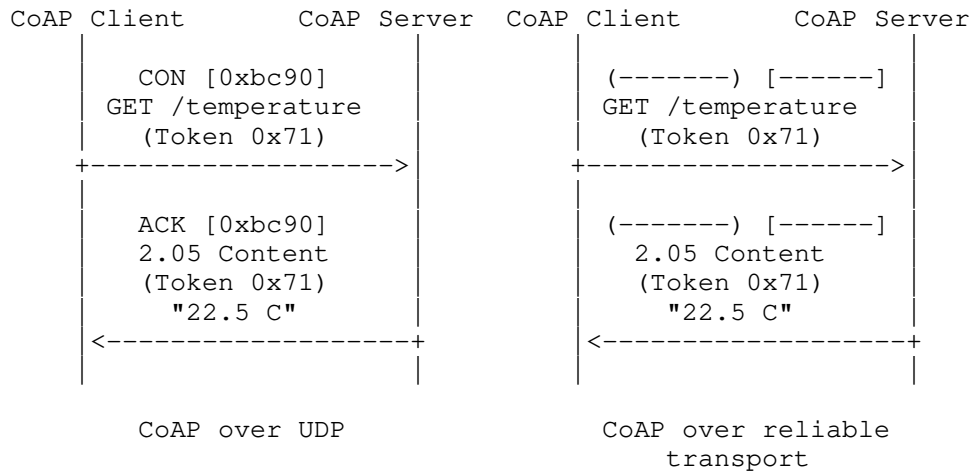


Figure 2: Comparison between CoAP over unreliable and reliable transport

3.2. Message Format

The CoAP message format defined in [RFC7252], as shown in Figure 3, relies on the datagram transport (UDP, or DTLS over UDP) for keeping the individual messages separate and for providing length information.

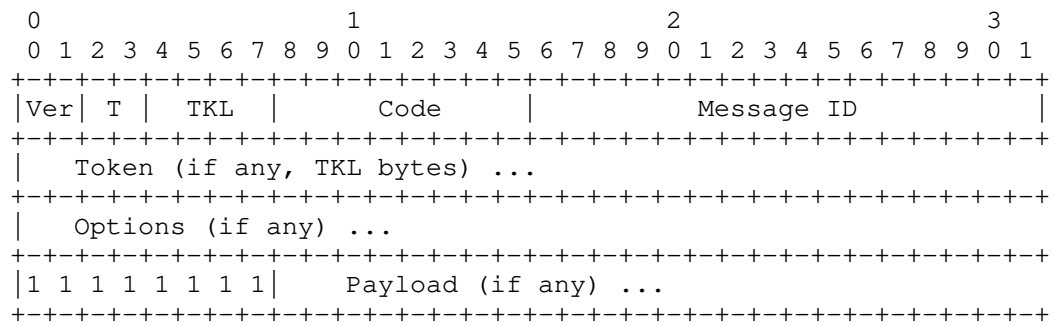


Figure 3: RFC 7252 defined CoAP Message Format

The CoAP over TCP message format is very similar to the format specified for CoAP over UDP. The differences are as follows:

- o Since the underlying TCP connection provides retransmissions and deduplication, there is no need for the reliability mechanisms provided by CoAP over UDP. The Type (T) and Message ID fields in the CoAP message header are elided.
- o The Version (Vers) field is elided as well. In contrast to the message format of CoAP over UDP, the message format for CoAP over TCP does not include a version number. CoAP is defined in [RFC7252] with a version number of 1. At this time, there is no known reason to support version numbers different from 1. If version negotiation needs to be addressed in the future, then Capabilities and Settings Messages (CSM see Section 5.3) have been specifically designed to enable such a potential feature.
- o In a stream oriented transport protocol such as TCP, a form of message delimitation is needed. For this purpose, CoAP over TCP introduces a length field with variable size. Figure 4 shows the adjusted CoAP message format with a modified structure for the fixed header (first 4 bytes of the CoAP over UDP header), which includes the length information of variable size.

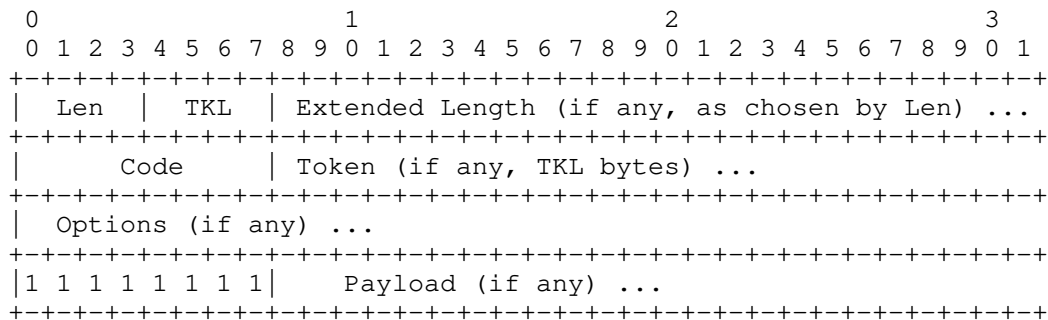


Figure 4: CoAP frame for reliable transports

Length (Len): 4-bit unsigned integer. A value between 0 and 12 inclusive indicates the length of the message in bytes starting with the first bit of the Options field. Three values are reserved for special constructs:

- 13: An 8-bit unsigned integer (Extended Length) follows the initial byte and indicates the length of options/payload minus 13.

14: A 16-bit unsigned integer (Extended Length) in network byte order follows the initial byte and indicates the length of options/payload minus 269.

15: A 32-bit unsigned integer (Extended Length) in network byte order follows the initial byte and indicates the length of options/payload minus 65805.

The encoding of the Length field is modeled after the Option Length field of the CoAP Options (see Section 3.1 of [RFC7252]).

For simplicity, a Payload Marker (0xFF) is shown in Figure 4; the Payload Marker indicates the start of the optional payload and is absent for zero-length payloads (see Section 3 of [RFC7252]). (If present, the Payload Marker is included in the message length, which counts from the start of the Options field to the end of the Payload field.)

For example: A CoAP message just containing a 2.03 code with the token 7f and no options or payload is encoded as shown in Figure 5.

```

      0               1               2
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           0x01           |           0x43           |           0x7f           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

Len   =    0 -----> 0x01
TKL   =    1 ____/
Code  =  2.03      --> 0x43
Token =                0x7f

```

Figure 5: CoAP message with no options or payload

The semantics of the other CoAP header fields are left unchanged.

3.3. Message Transmission

Once a transport connection is established, each endpoint MUST send a Capabilities and Settings message (CSM, see Section 5.3) as their first message on the connection. This message establishes the initial settings and capabilities for the endpoint, such as maximum message size or support for block-wise transfers. The absence of options in the CSM indicates that base values are assumed.

To avoid a deadlock, the Connection Initiator MUST NOT wait for the Connection Acceptor to send its initial CSM message before sending its own initial CSM message. Conversely, the Connection Acceptor MAY

wait for the Connection Initiator to send its initial CSM message before sending its own initial CSM message.

To avoid unnecessary latency, a Connection Initiator MAY send additional messages after its initial CSM without waiting to receive the Connection Acceptor's CSM; however, it is important to note that the Connection Acceptor's CSM might indicate capabilities that impact how the initiator is expected to communicate with the acceptor. For example, the acceptor CSM could indicate a Max-Message-Size option (see Section 5.3.1) that is smaller than the base value (1152) in order to limit both buffering requirements and head-of-line blocking.

Endpoints MUST treat a missing or invalid CSM as a connection error and abort the connection (see Section 5.6).

CoAP requests and responses are exchanged asynchronously over the transport connection. A CoAP client can send multiple requests without waiting for a response and the CoAP server can return responses in any order. Responses MUST be returned over the same connection as the originating request. Concurrent requests are differentiated by their Token, which is scoped locally to the connection.

The transport connection is bi-directional, so requests can be sent both by the entity that established the connection (Connection Initiator) and the remote host (Connection Acceptor). If one side does not implement a CoAP server, an error response MUST be returned for all CoAP requests from the other side. The simplest approach is to always return 5.01 (Not Implemented). A more elaborate mock server could also return 4.xx responses such as 4.04 (Not Found) or 4.02 (Bad Option) where appropriate.

Retransmission and deduplication of messages is provided by the TCP protocol.

3.4. Connection Health

Empty messages (Code 0.00) can always be sent and MUST be ignored by the recipient. This provides a basic keep-alive function that can refresh NAT bindings.

If a CoAP client does not receive any response for some time after sending a CoAP request (or, similarly, when a client observes a resource and it does not receive any notification for some time), it can send a CoAP Ping Signaling message (see Section 5.4) to test the transport connection and verify that the CoAP server is responsive.

When the underlying transport connection is closed or reset, the signaling state and any observation state (see Section 7.4) associated with the connection are removed. In flight messages may or may not be lost.

4. CoAP over WebSockets

CoAP over WebSockets is intentionally similar to CoAP over TCP; therefore, this section only specifies the differences between the transports.

CoAP over WebSockets can be used in a number of configurations. The most basic configuration is a CoAP client retrieving or updating a CoAP resource located on a CoAP server that exposes a WebSocket endpoint (see Figure 6). The CoAP client acts as the WebSocket client, establishes a WebSocket connection, and sends a CoAP request, to which the CoAP server returns a CoAP response. The WebSocket connection can be used for any number of requests.

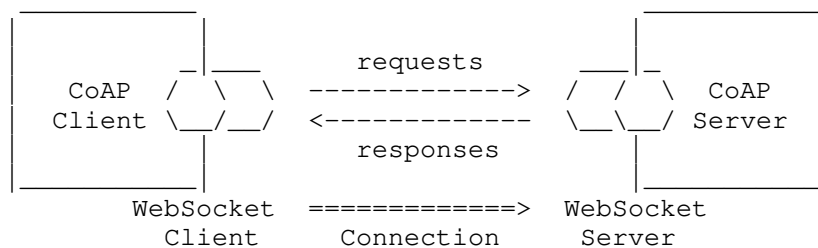


Figure 6: CoAP Client (WebSocket client) accesses CoAP Server (WebSocket server)

The challenge with this configuration is how to identify a resource in the namespace of the CoAP server. When the WebSocket protocol is used by a dedicated client directly (i.e., not from a web page through a web browser), the client can connect to any WebSocket endpoint. Section 8.3 and Section 8.4 define new URI schemes that enable the client to identify both a WebSocket endpoint and the path and query of the CoAP resource within that endpoint.

Another possible configuration is to set up a CoAP forward proxy at the WebSocket endpoint. Depending on what transports are available to the proxy, it could forward the request to a CoAP server with a CoAP UDP endpoint (Figure 7), an SMS endpoint (a.k.a. mobile phone), or even another WebSocket endpoint. The CoAP client specifies the resource to be updated or retrieved in the Proxy-Uri Option.

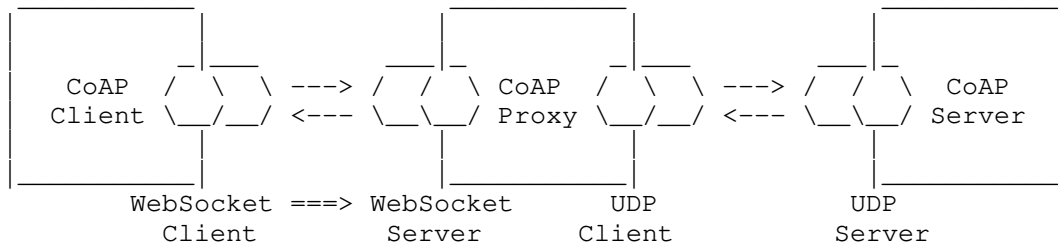


Figure 7: CoAP Client (WebSocket client) accesses CoAP Server (UDP server) via a CoAP proxy (WebSocket server/UDP client)

A third possible configuration is a CoAP server running inside a web browser (Figure 8). The web browser initially connects to a WebSocket endpoint and is then reachable through the WebSocket server. When no connection exists, the CoAP server is unreachable. Because the WebSocket server is the only way to reach the CoAP server, the CoAP proxy should be a reverse-proxy.

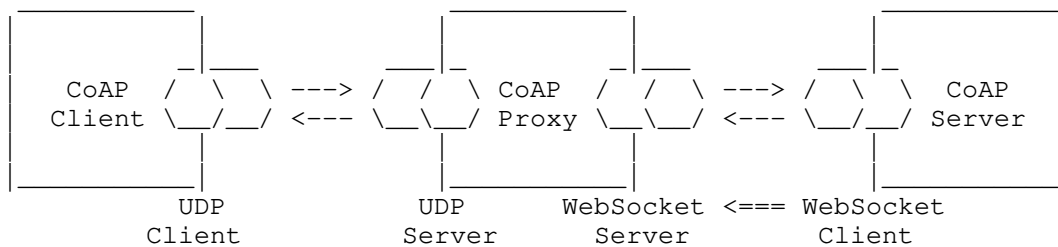


Figure 8: CoAP Client (UDP client) accesses CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

Further configurations are possible, including those where a WebSocket connection is established through an HTTP proxy.

4.1. Opening Handshake

Before CoAP requests and responses are exchanged, a WebSocket connection is established as defined in Section 4 of [RFC6455]. Figure 9 shows an example.

The WebSocket client MUST include the subprotocol name "coap" in the list of protocols, which indicates support for the protocol defined in this document.

The WebSocket client includes the hostname of the WebSocket server in the Host header field of its handshake as per [RFC6455]. The Host

header field also indicates the default value of the Uri-Host Option in requests from the WebSocket client to the WebSocket server.

```
GET /.well-known/coap HTTP/1.1
Host: example.org
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Protocol: coap
Sec-WebSocket-Version: 13

HTTP/1.1 101 Switching Protocols
Upgrade: websocket
Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
Sec-WebSocket-Protocol: coap
```

Figure 9: Example of an Opening Handshake

4.2. Message Format

Once a WebSocket connection is established, CoAP requests and responses can be exchanged as WebSocket messages. Since CoAP uses a binary message format, the messages are transmitted in binary data frames as specified in Sections 5 and 6 of [RFC6455].

The message format shown in Figure 10 is the same as the CoAP over TCP message format (see Section 3.2) with one change. The Length (Len) field MUST be set to zero because the WebSockets frame contains the length.

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Len=0 | TKL | Code | Token (TKL bytes) ... |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| Options (if any) ... |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
| 1 1 1 1 1 1 1 1 | Payload (if any) ... |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Figure 10: CoAP Message Format over WebSockets

As with CoAP over TCP, the message format for CoAP over WebSockets eliminates the Version field defined in CoAP over UDP. If CoAP version negotiation is required in the future, CoAP over WebSockets can address the requirement by the definition of a new subprotocol identifier that is negotiated during the opening handshake.

Requests and response messages can be fragmented as specified in Section 5.4 of [RFC6455], though typically they are sent unfragmented as they tend to be small and fully buffered before transmission. The WebSocket protocol does not provide means for multiplexing. If it is not desirable for a large message to monopolize the connection, requests and responses can be transferred in a block-wise fashion as defined in [RFC7959].

4.3. Message Transmission

As with CoAP over TCP, each endpoint MUST send a Capabilities and Settings message (CSM see Section 5.3) as their first message on the WebSocket connection.

CoAP requests and responses are exchanged asynchronously over the WebSocket connection. A CoAP client can send multiple requests without waiting for a response and the CoAP server can return responses in any order. Responses MUST be returned over the same connection as the originating request. Concurrent requests are differentiated by their Token, which is scoped locally to the connection.

The connection is bi-directional, so requests can be sent both by the entity that established the connection and the remote host.

As with CoAP over TCP, retransmission and deduplication of messages is provided by the WebSocket protocol. CoAP over WebSockets therefore does not make a distinction between Confirmable or Non-Confirmable messages, and does not provide Acknowledgement or Reset messages.

4.4. Connection Health

As with CoAP over TCP, a CoAP client can test the health of the CoAP over WebSocket connection by sending a CoAP Ping Signaling message (Section 5.4). WebSocket Ping and unsolicited Pong frames (Section 5.5 of [RFC6455]) SHOULD NOT be used to ensure that redundant maintenance traffic is not transmitted.

5. Signaling

Signaling messages are specifically introduced only for CoAP over reliable transports to allow peers to:

- o Learn related characteristics, such as maximum message size for the connection
- o Shut down the connection in an orderly fashion

- o Provide diagnostic information when terminating a connection in response to a serious error condition

Signaling is a third basic kind of message in CoAP, after requests and responses. Signaling messages share a common structure with the existing CoAP messages. There is a code, a token, options, and an optional payload.

(See Section 3 of [RFC7252] for the overall structure of the message format, option format, and option value format.)

5.1. Signaling Codes

A code in the 7.00-7.31 range indicates a Signaling message. Values in this range are assigned by the "CoAP Signaling Codes" sub-registry (see Section 11.1).

For each message, there is a sender and a peer receiving the message.

Payloads in Signaling messages are diagnostic payloads as defined in Section 5.5.2 of [RFC7252]), unless otherwise defined by a Signaling message option.

5.2. Signaling Option Numbers

Option numbers for Signaling messages are specific to the message code. They do not share the number space with CoAP options for request/response messages or with Signaling messages using other codes.

Option numbers are assigned by the "CoAP Signaling Option Numbers" sub-registry (see Section 11.2).

Signaling options are elective or critical as defined in Section 5.4.1 of [RFC7252]. If a Signaling option is critical and not understood by the receiver, it MUST abort the connection (see Section 5.6). If the option is understood but cannot be processed, the option documents the behavior.

5.3. Capabilities and Settings Messages (CSM)

Capabilities and Settings messages (CSM) are used for two purposes:

- o Each capability option indicates one capability of the sender to the recipient.
- o Each setting option indicates a setting that will be applied by the sender.

One CSM MUST be sent by each endpoint at the start of the transport connection. Further CSM MAY be sent at any other time by either endpoint over the lifetime of the connection.

Both capability and setting options are cumulative. A CSM does not invalidate a previously sent capability indication or setting even if it is not repeated. A capability message without any option is a no-operation (and can be used as such). An option that is sent might override a previous value for the same option. The option defines how to handle this case if needed.

Base values are listed below for CSM Options. These are the values for the capability and setting before any Capabilities and Settings messages send a modified value.

These are not default values for the option, as defined in Section 5.4.4 in [RFC7252]. Default values apply on a per-message basis and thus reset when the value is not present in a given Capabilities and Settings message.

Capabilities and Settings messages are indicated by the 7.01 code (CSM).

5.3.1. Max-Message-Size Capability Option

The sender can use the elective Max-Message-Size Option to indicate the maximum size of a message in bytes that it can receive. The message size indicated includes the entire message, starting from the first byte of the message header and ending at the end of the message payload.

(Note that there is no relationship of the message size to the overall request or response body size that may be achievable in block-wise transfer. For example, the exchange depicted further down in Figure 13 can be performed if the CoAP client indicates a value of around 6000 bytes for the Max-Message-Size option, even though the total body size transferred to the client is $3072 + 5120 + 4711 = 12903$ bytes.)

#	C	R	Applies to	Name	Format	Length	Base Value
2			CSM	Max-Message-Size	uint	0-4	1152

C=Critical, R=Repeatable

As per Section 4.6 of [RFC7252], the base value (and the value used when this option is not implemented) is 1152.

The active value of the Max-Message-Size Option is replaced each time the option is sent with a modified value. Its starting value is its base value.

5.3.2. Block-Wise-Transfer Capability Option

#	C	R	Applies to	Name	Format	Length	Base Value
4			CSM	Block-Wise-Transfer	empty	0	(none)

C=Critical, R=Repeatable

A sender can use the elective Block-Wise-Transfer Option to indicate that it supports the block-wise transfer protocol [RFC7959].

If the option is not given, the peer has no information about whether block-wise transfers are supported by the sender or not. An implementation wishing to offer block-wise transfers to its peer therefore needs to indicate the Block-Wise-Transfer Option.

If a Max-Message-Size Option is indicated with a value that is greater than 1152 (in the same or a different CSM message), the Block-Wise-Transfer Option also indicates support for BERT (see Section 6). Subsequently, if the Max-Message-Size Option is indicated with a value equal to or less than 1152, BERT support is no longer indicated. (Note that indication of BERT support obliges neither peer to actually choose to make use of BERT.)

Implementation note: When indicating a value of the Max-Message-Size option with an intention to enable BERT, the indicating implementation may want to choose a BERT size message it wants to encourage and add a delta for the header and any options that also need to be included in the message. Section 4.6 of [RFC7252] adds 128 bytes to a maximum block size of 1024 to arrive at a default message size of 1152. A BERT-enabled implementation may want to indicate a BERT block size of 2048 or a higher multiple of 1024, and at the same time be more generous for the size of header and options added (say, 256 or 512). Adding 1024 or more however to the base BERT block size may encourage the peer implementation to vary the BERT block size based on the size of the options included, which can be harder to establish interoperability for.

5.4. Ping and Pong Messages

In CoAP over reliable transports, Empty messages (Code 0.00) can always be sent and MUST be ignored by the recipient. This provides a basic keep-alive function. In contrast, Ping and Pong messages are a bidirectional exchange.

Upon receipt of a Ping message, the receiver MUST return a Pong message with an identical token in response. Unless the Ping carries an option with delaying semantics such as the Custody Option, it SHOULD respond as soon as practical. As with all Signaling messages, the recipient of a Ping or Pong message MUST ignore elective options it does not understand.

Ping and Pong messages are indicated by the 7.02 code (Ping) and the 7.03 code (Pong).

Note that, as with similar mechanisms defined in [RFC6455] and [RFC7540], the present specification does not define any specific maximum time that the sender of a Ping message has to allow waiting for a Pong reply. Any limitations on the patience for this reply are a matter of the application making use of these messages, as is any approach to recover from a failure to respond in time.

5.4.1. Custody Option

#	C	R	Applies to	Name	Format	Length	Base Value
2			Ping, Pong	Custody	empty	0	(none)

C=Critical, R=Repeatable

When responding to a Ping message, the receiver can include an elective Custody Option in the Pong message. This option indicates that the application has processed all the request/response messages received prior to the Ping message on the current connection. (Note that there is no definition of specific application semantics for "processed", but there is an expectation that the receiver of a Pong Message with a Custody Option should be able to free buffers based on this indication.)

A sender can also include an elective Custody Option in a Ping message to explicitly request the inclusion of an elective Custody Option in the corresponding Pong message. In that case, the receiver

SHOULD delay its Pong message until it finishes processing all the request/response messages received prior to the Ping message on the current connection.

5.5. Release Messages

A Release message indicates that the sender does not want to continue maintaining the transport connection and opts for an orderly shutdown, but wants to leave it to the peer to actually start closing the connection. The details are in the options. A diagnostic payload (see Section 5.5.2 of [RFC7252]) MAY be included.

A peer will normally respond to a Release message by closing the transport connection. (In case that does not happen, the sender of the release may want to implement a timeout mechanism if getting rid of the connection is actually important to it.)

Messages may be in flight or responses outstanding when the sender decides to send a Release message (which is one reason the sender had decided to wait with closing the connection). The peer responding to the Release message SHOULD delay the closing of the connection until it has responded to all requests received by it before the Release message. It also MAY wait for the responses to its own requests.

It is NOT RECOMMENDED for the sender of a Release message to continue sending requests on the connection it already indicated to be released: the peer might close the connection at any time and miss those requests. There is no obligation for the peer to check for this condition, though.

Release messages are indicated by the 7.04 code (Release).

Release messages can indicate one or more reasons using elective options. The following options are defined:

#	C	R	Applies to	Name	Format	Length	Base Value
2		x	Release	Alternative-Address	string	1-255	(none)

C=Critical, R=Repeatable

The elective Alternative-Address Option requests the peer to instead open a connection of the same scheme as the present connection to the alternative transport address given. Its value is in the form

"authority" as defined in Section 3.2 of [RFC3986]. (Existing state related to the connection is not transferred from the present connection to the new connection.)

The Alternative-Address Option is a repeatable option as defined in Section 5.4.5 of [RFC7252]. When multiple occurrences of the option are included, the peer can choose any of the alternative transport addresses.

#	C	R	Applies to	Name	Format	Length	Base Value
4			Release	Hold-Off	uint	0-3	(none)

C=Critical, R=Repeatable

The elective Hold-Off Option indicates that the server is requesting that the peer not reconnect to it for the number of seconds given in the value.

5.6. Abort Messages

An Abort message indicates that the sender is unable to continue maintaining the transport connection and cannot even wait for an orderly release. The sender shuts down the connection immediately after the abort (and may or may not wait for a Release or Abort message or connection shutdown in the inverse direction). A diagnostic payload (see Section 5.5.2 of [RFC7252]) SHOULD be included in the Abort message. Messages may be in flight or responses outstanding when the sender decides to send an Abort message. The general expectation is that these will NOT be processed.

Abort messages are indicated by the 7.05 code (Abort).

Abort messages can indicate one or more reasons using elective options. The following option is defined:

#	C	R	Applies to	Name	Format	Length	Base Value
2			Abort	Bad-CSM-Option	uint	0-2	(none)

C=Critical, R=Repeatable

The elective Bad-CSM-Option Option indicates that the sender is unable to process the CSM option identified by its option number, e.g. when it is critical and the option number is unknown by the sender, or when there is parameter problem with the value of an elective option. More detailed information SHOULD be included as a diagnostic payload.

For CoAP over UDP, messages which contain syntax violations are processed as message format errors. As described in Sections 4.2 and 4.3 of [RFC7252], such messages are rejected by sending a matching Reset message and otherwise ignoring the message.

For CoAP over reliable transports, the recipient rejects such messages by sending an Abort message and otherwise ignoring (not processing) the message. No specific option has been defined for the Abort message in this case, as the details are best left to a diagnostic payload.

5.7. Signaling examples

An encoded example of a Ping message with a non-empty token is shown in Figure 11.

```

      0                               1                               2
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           0x01           |           0xe2           |           0x42           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

Len    =    0 -----> 0x01
TKL    =    1 ____/
Code   = 7.02 Ping --> 0xe2
Token  =                               0x42

```

Figure 11: Ping Message Example

An encoded example of the corresponding Pong message is shown in Figure 12.

```

      0               1               2
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           0x01           |           0xe3           |           0x42           |
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+

Len   =   0 -----> 0x01
TKL   =   1 ____/
Code  = 7.03 Pong --> 0xe3
Token =                   0x42

```

Figure 12: Pong Message Example

6. Block-wise Transfer and Reliable Transports

The message size restrictions defined in Section 4.6 of CoAP [RFC7252] to avoid IP fragmentation are not necessary when CoAP is used over a reliable transport. While this suggests that the Block-wise transfer protocol [RFC7959] is also no longer needed, it remains applicable for a number of cases:

- o large messages, such as firmware downloads, may cause undesired head-of-line blocking when a single transport connection is used
- o a UDP-to-TCP gateway may simply not have the context to convert a message with a Block Option into the equivalent exchange without any use of a Block Option (it would need to convert the entire blockwise exchange from start to end into a single exchange)

The 'Block-wise Extension for Reliable Transport (BERT)' extends the Block protocol to enable the use of larger messages over a reliable transport.

The use of this new extension is signaled by sending Block1 or Block2 Options with SZX == 7 (a "BERT option"). SZX == 7 is a reserved value in [RFC7959].

In control usage, a BERT option is interpreted in the same way as the equivalent Option with SZX == 6, except that it also indicates the capability to process BERT blocks. As with the basic Block protocol, the recipient of a CoAP request with a BERT option in control usage is allowed to respond with a different SZX value, e.g. to send a non-BERT block instead.

In descriptive usage, a BERT Option is interpreted in the same way as the equivalent Option with SZX == 6, except that the payload is also allowed to contain multiple blocks. For non-final BERT blocks, the payload is always a multiple of 1024 bytes. For final BERT blocks,

the payload is a multiple (possibly 0) of 1024 bytes plus a partial block of less than 1024 bytes.

The recipient of a non-final BERT block ($M=1$) conceptually partitions the payload into a sequence of 1024-byte blocks and acts exactly as if it had received this sequence in conjunction with block numbers starting at, and sequentially increasing from, the block number given in the Block Option. In other words, the entire BERT block is positioned at the byte position that results from multiplying the block number with 1024. The position of further blocks to be transferred is indicated by incrementing the block number by the number of elements in this sequence (i.e., the size of the payload divided by 1024 bytes).

As with $SZX == 6$, the recipient of a final BERT block ($M=0$) simply appends the payload at the byte position that is indicated by the block number multiplied with 1024.

The following examples illustrate BERT options. A value of $SZX == 7$ is labeled as "BERT" or as "BERT(nnn)" to indicate a payload of size nnn.

In all these examples, a Block Option is decomposed to indicate the kind of Block Option (1 or 2) followed by a colon, the block number (NUM), more bit (M), and block size ($2^{*(SZX+4)}$) separated by slashes. E.g., a Block2 Option value of 33 would be shown as 2:2/0/32), or a Block1 Option value of 59 would be shown as 1:3/1/128.

6.1. Example: GET with BERT Blocks

Figure 13 shows a GET request with a response that is split into three BERT blocks. The first response contains 3072 bytes of payload; the second, 5120; and the third, 4711. Note how the block number increments to move the position inside the response body forward.

CoAP Client	CoAP Server
GET, /status	----->
<----- 2.05 Content, 2:0/1/BERT(3072)	
GET, /status, 2:3/0/BERT	----->
<----- 2.05 Content, 2:3/1/BERT(5120)	
GET, /status, 2:8/0/BERT	----->
<----- 2.05 Content, 2:8/0/BERT(4711)	

Figure 13: GET with BERT blocks

6.2. Example: PUT with BERT Blocks

Figure 14 demonstrates a PUT exchange with BERT blocks.

CoAP Client	CoAP Server
PUT, /options, 1:0/1/BERT(8192)	----->
<----- 2.31 Continue, 1:0/1/BERT	
PUT, /options, 1:8/1/BERT(16384)	----->
<----- 2.31 Continue, 1:8/1/BERT	
PUT, /options, 1:24/0/BERT(5683)	----->
<----- 2.04 Changed, 1:24/0/BERT	

Figure 14: PUT with BERT blocks

7. Observing Resources over Reliable Transports

This section describes how the procedures defined in [RFC7641] for observing resources over CoAP are applied (and modified, as needed) for reliable transports. In this section, "client" and "server" refer to the CoAP client and CoAP server.

7.1. Notifications and Reordering

When using the Observe Option with CoAP over UDP, notifications from the server set the option value to an increasing sequence number for reordering detection on the client since messages can arrive in a different order than they were sent. This sequence number is not required for CoAP over reliable transports since the TCP protocol ensures reliable and ordered delivery of messages. The value of the Observe Option in 2.xx notifications MAY be empty on transmission and MUST be ignored on reception.

Implementation note: This means that a proxy from a reordering transport to a reliable (in-order) transport (such as a UDP-to-TCP proxy) needs to process the Observe Option in notifications according to the rules in Section 3.4 of [RFC7641].

7.2. Transmission and Acknowledgements

For CoAP over UDP, server notifications to the client can be confirmable or non-confirmable. A confirmable message requires the client to either respond with an acknowledgement message or a reset message. An acknowledgement message indicates that the client is alive and wishes to receive further notifications. A reset message indicates that the client does not recognize the token which causes the server to remove the associated entry from the list of observers.

Since TCP eliminates the need for the message layer to support reliability, CoAP over reliable transports does not support confirmable or non-confirmable message types. All notifications are delivered reliably to the client with positive acknowledgement of receipt occurring at the TCP level. If the client does not recognize the token in a notification, it MAY immediately abort the connection (see Section 5.6).

7.3. Freshness

For CoAP over UDP, if a client does not receive a notification for some time, it MAY send a new GET request with the same token as the original request to re-register its interest in a resource and verify that the server is still responsive. For CoAP over reliable transports, it is more efficient to check the health of the connection (and all its active observations) by sending a single CoAP Ping Signaling message (Section 5.4) rather than individual requests to confirm each active observation. (Note that such a Ping/Pong only confirms a single hop: there is no obligation, and no expectation, of a proxy to react to a Ping by checking all its onward observations or all the connections, if any, underlying them. A proxy MAY maintain its own schedule for confirming the onward observations it relies on;

it is however generally inadvisable for a proxy to generate a large number of outgoing checks based on a single incoming check.)

7.4. Cancellation

For CoAP over UDP, a client that is no longer interested in receiving notifications can "forget" the observation and respond to the next notification from the server with a reset message to cancel the observation.

For CoAP over reliable transports, a client MUST explicitly deregister by issuing a GET request that has the Token field set to the token of the observation to be cancelled and includes an Observe Option with the value set to 1 (deregister).

If the client observes one or more resources over a reliable transport, then the CoAP server (or intermediary in the role of the CoAP server) MUST remove all entries associated with the client endpoint from the lists of observers when the connection is either closed or times out.

8. CoAP over Reliable Transport URIs

CoAP over UDP [RFC7252] defines the "coap" and "coaps" URI schemes. This document introduces four additional URI schemes for identifying CoAP resources and providing a means of locating the resource:

- o the "coap+tcp" URI scheme for CoAP over TCP
- o the "coaps+tcp" URI scheme for CoAP over TCP secured by TLS
- o the "coap+ws" URI scheme for CoAP over WebSockets
- o the "coaps+ws" URI scheme for CoAP over WebSockets secured by TLS

Resources made available via these schemes have no shared identity even if their resource identifiers indicate the same authority (the same host listening to the same TCP port). They are hosted in distinct namespaces because each URI scheme implies a distinct origin server.

The syntax for the URI schemes in this section are specified using Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty", and "query" are adopted from [RFC3986].

Section 8 (Multicast CoAP) in [RFC7252] is not applicable to these schemes.

As with the "coap" and "coaps" schemes defined in [RFC7252], all URI schemes defined in this section also support the path prefix `"/.well-known/"` defined by [RFC5785] for "well-known locations" in the namespace of a host. This enables discovery as per Section 7 of [RFC7252].

8.1. coap+tcp URI scheme

The "coap+tcp" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over TCP.

```
coap-tcp-URI = "coap+tcp:" "/" host [ ":" port ]
               path-abempty [ "?" query ]
```

The syntax defined in Section 6.1 of [RFC7252] applies to this URI scheme with the following changes:

- o The port subcomponent indicates the TCP port at which the CoAP Connection Acceptor is located. (If it is empty or not given, then the default port 5683 is assumed, as with UDP.)

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.2. coaps+tcp URI scheme

The "coaps+tcp" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over TCP secured with TLS.

```
coaps-tcp-URI = "coaps+tcp:" "/" host [ ":" port ]
                path-abempty [ "?" query ]
```

The syntax defined in Section 6.2 of [RFC7252] applies to this URI scheme, with the following changes:

- o The port subcomponent indicates the TCP port at which the TLS server for the CoAP Connection Acceptor is located. If it is empty or not given, then the default port 5684 is assumed.
- o If a TLS server does not support the Application-Layer Protocol Negotiation Extension (ALPN) [RFC7301] or wishes to accommodate TLS clients that do not support ALPN, it MAY offer a coaps+tcp endpoint on TCP port 5684. This endpoint MAY also be ALPN

enabled. A TLS server MAY offer coaps+tcp endpoints on ports other than TCP port 5684, which MUST be ALPN enabled.

- o For TCP ports other than port 5684, the TLS client MUST use the ALPN extension to advertise the "coap" protocol identifier (see Section 11.7) in the list of protocols in its ClientHello. If the TCP server selects and returns the "coap" protocol identifier using the ALPN extension in its ServerHello, then the connection succeeds. If the TLS server either does not negotiate the ALPN extension or returns a no_application_protocol alert, the TLS client MUST close the connection.
- o For TCP port 5684, a TLS client MAY use the ALPN extension to advertise the "coap" protocol identifier in the list of protocols in its ClientHello. If the TLS server selects and returns the "coap" protocol identifier using the ALPN extension in its ServerHello, then the connection succeeds. If the TLS server returns a no_application_protocol alert, then the TLS client MUST close the connection. If the TLS server does not negotiate the ALPN extension, then coaps+tcp is implicitly selected.
- o For TCP port 5684, if the TLS client does not use the ALPN extension to negotiate the protocol, then coaps+tcp is implicitly selected.

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.3. coap+ws URI scheme

The "coap+ws" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over WebSockets.

```
coap-ws-URI = "coap+ws:" "/" host [ ":" port ]
              path-abempty [ "?" query ]
```

The port subcomponent is OPTIONAL. The default is port 80.

The WebSocket endpoint is identified by a "ws" URI that is composed of the authority part of the "coap+ws" URI and the well-known path `"/.well-known/coap"` [RFC5785] [I-D.bormann-hybi-ws-wk]. The path and query parts of a "coap+ws" URI identify a resource within the specified endpoint which can be operated on by the methods defined by CoAP:

```

coap+ws://example.org/sensors/temperature?u=Cel
      \_____/ \_____/ \_____/
      \_/      \_/      \_/
ws://example.org/.well-known/coap  Uri-Path: "sensors"
                                   Uri-Path: "temperature"
                                   Uri-Query: "u=Cel"

```

Figure 15: The "coap+ws" URI Scheme

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.4. coaps+ws URI scheme

The "coaps+ws" URI scheme identifies CoAP resources that are intended to be accessible using CoAP over WebSockets secured by TLS.

```

coaps-ws-URI = "coaps+ws:" "/" host [ ":" port ]
path-abempty [ "?" query ]

```

The port subcomponent is OPTIONAL. The default is port 443.

The WebSocket endpoint is identified by a "wss" URI that is composed of the authority part of the "coaps+ws" URI and the well-known path `"/.well-known/coap"` [RFC5785] [I-D.bormann-hybi-ws-wk]. The path and query parts of a "coaps+ws" URI identify a resource within the specified endpoint which can be operated on by the methods defined by CoAP.

```

coaps+ws://example.org/sensors/temperature?u=Cel
      \_____/ \_____/ \_____/
      \_/      \_/      \_/
wss://example.org/.well-known/coap  Uri-Path: "sensors"
                                   Uri-Path: "temperature"
                                   Uri-Query: "u=Cel"

```

Figure 16: The "coaps+ws" URI Scheme

Encoding considerations: The scheme encoding conforms to the encoding rules established for URIs in [RFC3986].

Interoperability considerations: None.

Security considerations: See Section 11.1 of [RFC7252].

8.5. Uri-Host and Uri-Port Options

CoAP over reliable transports maintains the property from Section 5.10.1 of [RFC7252]:

The default values for the Uri-Host and Uri-Port Options are sufficient for requests to most servers.

Unless otherwise noted, the default value of the Uri-Host Option is the IP literal representing the destination IP address of the request message. The default value of the Uri-Port Option is the destination TCP port.

For CoAP over TLS, these default values are the same unless Server Name Indication (SNI) [RFC6066] is negotiated. In this case, the default value of the Uri-Host Option in requests from the TLS client to the TLS server is the SNI host.

For CoAP over WebSockets, the default value of the Uri-Host Option in requests from the WebSocket client to the WebSocket server is indicated by the Host header field from the WebSocket handshake.

8.6. Decomposing URIs into Options

The steps are the same as specified in Section 6.4 of [RFC7252] with minor changes.

This step from [RFC7252]:

3. If `|url|` does not have a `<scheme>` component whose value, when converted to ASCII lowercase, is "coap" or "coaps", then fail this algorithm.

is updated to:

3. If `|url|` does not have a `<scheme>` component whose value, when converted to ASCII lowercase, is "coap+tcp", "coaps+tcp", "coap+ws", or "coaps+ws", then fail this algorithm.

This step from [RFC7252]:

7. If `|port|` does not equal the request's destination UDP port, include a Uri-Port Option and let that option's value be `|port|`.

is updated to:

7. If `|port|` does not equal the request's destination TCP port, include a Uri-Port Option and let that option's value be `|port|`.

8.7. Composing URIs from Options

The steps are the same as specified in Section 6.5 of [RFC7252] with minor changes.

This step from [RFC7252]:

1. If the request is secured using DTLS, let `|url|` be the string "coaps://". Otherwise, let `|url|` be the string "coap://".

is updated to:

1. For CoAP over TCP, if the request is secured using TLS, let `|url|` be the string "coaps+tcp://". Otherwise, let `|url|` be the string "coap+tcp://". For CoAP over WebSockets, if the request is secured using TLS, let `|url|` be the string "coaps+ws://". Otherwise, let `|url|` be the string "coap+ws://".

This step from [RFC7252]:

4. If the request includes a Uri-Port Option, let `|port|` be that option's value. Otherwise, let `|port|` be the request's destination UDP port.

is updated to:

4. If the request includes a Uri-Port Option, let `|port|` be that option's value. Otherwise, let `|port|` be the request's destination TCP port.

9. Securing CoAP

Security Challenges for the Internet of Things [SecurityChallenges] recommends:

... it is essential that IoT protocol suites specify a mandatory to implement but optional to use security solution. This will ensure security is available in all implementations, but configurable to use when not necessary (e.g., in closed environment). ... even if those features stretch the capabilities of such devices.

A security solution **MUST** be implemented to protect CoAP over reliable transports and **MUST** be enabled by default. This document defines the TLS binding, but alternative solutions at different layers in the protocol stack **MAY** be used to protect CoAP over reliable transports when appropriate. Note that there is ongoing work to support a data

object-based security model for CoAP that is independent of transport (see [I-D.ietf-core-object-security]).

9.1. TLS binding for CoAP over TCP

The TLS usage guidance in [RFC7925] applies, including the guidance about cipher suites in that document that are derived from the mandatory-to-implement (MTI) cipher suites defined in [RFC7525].

This guidance assumes implementation in a constrained device or for communication with a constrained device. CoAP over TCP/TLS has, however, a wider applicability. It may, for example, be implemented on a gateway or on a device that is less constrained (such as a smart phone or a tablet), for communication with a peer that is likewise less constrained, or within a backend environment that only communicates with constrained devices via proxies. As an exception to the previous paragraph, in this case, the recommendations in [RFC7525] are more appropriate.

Since the guidance offered in [RFC7925] and [RFC7525] differs in terms of algorithms and credential types, it is assumed that a CoAP over TCP/TLS implementation that needs to support both cases implements the recommendations offered by both specifications.

During the provisioning phase, a CoAP device is provided with the security information that it needs, including keying materials, access control lists, and authorization servers. At the end of the provisioning phase, the device will be in one of four security modes:

NoSec: TLS is disabled.

PreSharedKey: TLS is enabled. The guidance in Section 4.2 of [RFC7925] applies.

RawPublicKey: TLS is enabled. The guidance in Section 4.3 of [RFC7925] applies.

Certificate: TLS is enabled. The guidance in Section 4.4 of [RFC7925] applies.

The "NoSec" mode is optional-to-implement. The system simply sends the packets over normal TCP which is indicated by the "coap+tcp" scheme and the TCP CoAP default port. The system is secured only by keeping attackers from being able to send or receive packets from the network with the CoAP nodes.

"PreSharedKey", "RawPublicKey", or "Certificate" is mandatory-to-implement for the TLS binding depending on the credential type used

with the device. These security modes are achieved using TLS and are indicated by the "coaps+tcp" scheme and TLS-secured CoAP default port.

9.2. TLS usage for CoAP over WebSockets

A CoAP client requesting a resource identified by a "coaps+ws" URI negotiates a secure WebSocket connection to a WebSocket server endpoint with a "wss" URI. This is described in Section 8.4.

The client MUST perform a TLS handshake after opening the connection to the server. The guidance in Section 4.1 of [RFC6455] applies. When a CoAP server exposes resources identified by a "coaps+ws" URI, the guidance in Section 4.4 of [RFC7925] applies towards mandatory-to-implement TLS functionality for certificates. For the server-side requirements in accepting incoming connections over a HTTPS (HTTP-over-TLS) port, the guidance in Section 4.2 of [RFC6455] applies.

Note that this formally inherits the mandatory-to-implement cipher suites defined in [RFC5246]. However, usually modern browsers implement more recent cipher suites that then are automatically picked up via the JavaScript WebSocket API. WebSocket Servers that provide Secure CoAP over WebSockets for the browser use case will need to follow the browser preferences and MUST follow [RFC7525].

10. Security Considerations

The security considerations of [RFC7252] apply. For CoAP over WebSockets and CoAP over TLS-secured WebSockets, the security considerations of [RFC6455] also apply.

10.1. Signaling Messages

The guidance given by an Alternative-Address Option cannot be followed blindly. In particular, a peer MUST NOT assume that a successful connection to the Alternative-Address inherits all the security properties of the current connection.

11. IANA Considerations

11.1. Signaling Codes

IANA is requested to create a third sub-registry for values of the Code field in the CoAP header (Section 12.1 of [RFC7252]). The name of this sub-registry is "CoAP Signaling Codes".

Each entry in the sub-registry must include the Signaling Code in the range 7.00-7.31, its name, and a reference to its documentation.

Initial entries in this sub-registry are as follows:

Code	Name	Reference
7.01	CSM	[RFCthis]
7.02	Ping	[RFCthis]
7.03	Pong	[RFCthis]
7.04	Release	[RFCthis]
7.05	Abort	[RFCthis]

Table 1: CoAP Signal Codes

All other Signaling Codes are Unassigned.

The IANA policy for future additions to this sub-registry is "IETF Review or IESG Approval" as described in [RFC8126].

11.2. CoAP Signaling Option Numbers Registry

IANA is requested to create a sub-registry for Options Numbers used in CoAP signaling options within the "CoRE Parameters" registry. The name of this sub-registry is "CoAP Signaling Option Numbers".

Each entry in the sub-registry must include one or more of the codes in the Signaling Codes subregistry (Section 11.1), the option number, the name of the option, and a reference to the option's documentation.

Initial entries in this sub-registry are as follows:

Applies to	Number	Name	Reference
7.01	2	Max-Message-Size	[RFCthis]
7.01	4	Block-Wise-Transfer	[RFCthis]
7.02, 7.03	2	Custody	[RFCthis]
7.04	2	Alternative-Address	[RFCthis]
7.04	4	Hold-Off	[RFCthis]
7.05	2	Bad-CSM-Option	[RFCthis]

Table 2: CoAP Signal Option Codes

The IANA policy for future additions to this sub-registry is based on number ranges for the option numbers, analogous to the policy defined in Section 12.2 of [RFC7252]. (The policy is analogous rather than identical because the structure of the subregistry includes an additional column; however, the value of this column has no influence on the policy.)

The documentation for a Signaling Option Number should specify the semantics of an option with that number, including the following properties:

- o Whether the option is critical or elective, as determined by the Option Number.
- o Whether the option is repeatable.
- o The format and length of the option's value.
- o The base value for the option, if any.

11.3. Service Name and Port Number Registration

IANA is requested to assign the port number 5683 and the service name "coap+tcp", in accordance with [RFC6335].

Service Name.
coap+tcp

Transport Protocol.
tcp

Assignee.

IESG <iesg@ietf.org>

Contact.

IETF Chair <chair@ietf.org>

Description.

Constrained Application Protocol (CoAP)

Reference.

[RFCthis]

Port Number.

5683

11.4. Secure Service Name and Port Number Registration

IANA is requested to assign the port number 5684 and the service name "coaps+tcp", in accordance with [RFC6335]. The port number is requested to address the exceptional case of TLS implementations that do not support the "Application-Layer Protocol Negotiation Extension" [RFC7301].

Service Name.

coaps+tcp

Transport Protocol.

tcp

Assignee.

IESG <iesg@ietf.org>

Contact.

IETF Chair <chair@ietf.org>

Description.

Constrained Application Protocol (CoAP)

Reference.

[RFC7301], [RFCthis]

Port Number.

5684

11.5. URI Scheme Registration

URI schemes are registered within the "Uniform Resource Identifier (URI) Schemes" registry maintained at [IANA.uri-schemes].

11.5.1. coap+tcp

IANA is requested to register the Uniform Resource Identifier (URI) scheme "coap+tcp". This registration request complies with [RFC7595].

Scheme name:
coap+tcp

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using TCP.

Contact:
IETF chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 8.1 in [RFCthis]

11.5.2. coaps+tcp

IANA is requested to register the Uniform Resource Identifier (URI) scheme "coaps+tcp". This registration request complies with [RFC7595].

Scheme name:
coaps+tcp

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using TLS.

Contact:
IETF chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 8.2 in [RFCthis]

11.5.3. coap+ws

IANA is requested to register the Uniform Resource Identifier (URI) scheme "coap+ws". This registration request complies with [RFC7595].

Scheme name:
coap+ws

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol.

Contact:
IETF chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

Reference:
Section 8.3 in [RFCthis]

11.5.4. coaps+ws

IANA is requested to register the Uniform Resource Identifier (URI) scheme "coaps+ws". This registration request complies with [RFC7595].

Scheme name:
coaps+ws

Status:
Permanent

Applications/protocols that use this scheme name:
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol secured with TLS.

Contact:
IETF chair <chair@ietf.org>

Change controller:
IESG <iesg@ietf.org>

References:
Section 8.4 in [RFCthis]

11.6. Well-Known URI Suffix Registration

IANA is requested to register the 'coap' well-known URI in the "Well-Known URIs" registry. This registration request complies with [RFC5785]:

URI Suffix.
coap

Change controller.
IETF

Specification document(s).
[RFCthis]

Related information.
None.

11.7. ALPN Protocol Identifier

IANA is requested to assign the following value in the registry "Application Layer Protocol Negotiation (ALPN) Protocol IDs" created by [RFC7301]. The "coap" string identifies CoAP when used over TLS.

Protocol.
CoAP

Identification Sequence.
0x63 0x6f 0x61 0x70 ("coap")

Reference.
[RFCthis]

11.8. WebSocket Subprotocol Registration

IANA is requested to register the WebSocket CoAP subprotocol under the "WebSocket Subprotocol Name Registry":

Subprotocol Identifier.
coap

Subprotocol Common Name.

Constrained Application Protocol (CoAP)

Subprotocol Definition.
[RFCthis]

11.9. CoAP Option Numbers Registry

IANA is requested to add [RFCthis] to the references for the following entries registered by [RFC7959] in the "CoAP Option Numbers" sub-registry defined by [RFC7252]:

Number	Name	Reference
23	Block2	RFC 7959, [RFCthis]
27	Block1	RFC 7959, [RFCthis]

Table 3: CoAP Option Numbers

12. References

12.1. Normative References

- [I-D.bormann-hybi-ws-wk]
Bormann, C., "Well-known URIs for the WebSocket Protocol", draft-bormann-hybi-ws-wk-00 (work in progress), May 2017.
- [RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <<https://www.rfc-editor.org/info/rfc793>>.
- [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>>.
- [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>>.

- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <<https://www.rfc-editor.org/info/rfc5785>>.
- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<https://www.rfc-editor.org/info/rfc6066>>.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <<https://www.rfc-editor.org/info/rfc6455>>.
- [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>>.
- [RFC7301] Friedl, S., Popov, A., Langley, A., and E. Stephan, "Transport Layer Security (TLS) Application-Layer Protocol Negotiation Extension", RFC 7301, DOI 10.17487/RFC7301, July 2014, <<https://www.rfc-editor.org/info/rfc7301>>.
- [RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <<https://www.rfc-editor.org/info/rfc7525>>.
- [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>>.
- [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>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.

- [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>>.
- [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>>.

12.2. Informative References

- [BK2015] Byrne, C. and J. Kleberg, "Advisory Guidelines for UDP Deployment", Proceedings draft-byrne-opsec-udp-advisory-00 (expired), 2015.
- [EK2016] Edeline, K., Kuehlewind, M., Trammell, B., Aben, E., and B. Donnet, "Using UDP for Internet Transport Evolution", Proceedings arXiv preprint 1612.07816, 2016.
- [HomeGateway] Eggert, L., "An experimental study of home gateway characteristics", Proceedings of the 10th annual conference on Internet measurement , 2010.
- [I-D.gomez-lwig-tcp-constrained-node-networks] Gomez, C., Crowcroft, J., and M. Scharf, "TCP over Constrained-Node Networks", draft-gomez-lwig-tcp-constrained-node-networks-03 (work in progress), June 2017.
- [I-D.ietf-core-cocoa] Bormann, C., Betzler, A., Gomez, C., and I. Demirkol, "CoAP Simple Congestion Control/Advanced", draft-ietf-core-cocoa-02 (work in progress), October 2017.
- [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-07 (work in progress), November 2017.
- [IANA.uri-schemes] IANA, "Uniform Resource Identifier (URI) Schemes", <<http://www.iana.org/assignments/uri-schemes>>.

- [LWM2M] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification Version 1.0", February 2017, <http://www.openmobilealliance.org/release/LightweightM2M/V1_0-20170208-A/OMA-TS-LightweightM2M-V1_0-20170208-A.pdf>.
- [RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <<https://www.rfc-editor.org/info/rfc768>>.
- [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>>.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, DOI 10.17487/RFC6335, August 2011, <<https://www.rfc-editor.org/info/rfc6335>>.
- [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>>.
- [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>>.
- [RFC7540] Belshe, M., Peon, R., and M. Thomson, Ed., "Hypertext Transfer Protocol Version 2 (HTTP/2)", RFC 7540, DOI 10.17487/RFC7540, May 2015, <<https://www.rfc-editor.org/info/rfc7540>>.
- [SecurityChallenges] Polk, T. and S. Turner, "Security Challenges for the Internet of Things", Interconnecting Smart Objects with the Internet / IAB Workshop , February 2011, <<http://www.iab.org/wp-content/IAB-uploads/2011/03/Turner.pdf>>.
- [SW2016] Swett, I., "QUIC Deployment Experience @Google", Proceedings <https://www.ietf.org/proceedings/96/slides/slides-96-quic-3.pdf>, 2016.

Appendix A. CoAP over WebSocket Examples

This section gives examples for the first two configurations discussed in Section 4.

An example of the process followed by a CoAP client to retrieve the representation of a resource identified by a "coap+ws" URI might be as follows. Figure 17 below illustrates the WebSocket and CoAP messages exchanged in detail.

1. The CoAP client obtains the URI <coap+ws://example.org/sensors/temperature?u=Cel>, for example, from a resource representation that it retrieved previously.
2. It establishes a WebSocket connection to the endpoint URI composed of the authority "example.org" and the well-known path "/.well-known/coap", <ws://example.org/.well-known/coap>.
3. CSM messages (Section 5.3) are exchanged (not shown for lack of space).
4. It sends a single-frame, masked, binary message containing a CoAP request. The request indicates the target resource with the Uri-Path ("sensors", "temperature") and Uri-Query ("u=Cel") options.
5. It waits for the server to return a response.
6. The CoAP client uses the connection for further requests, or the connection is closed.

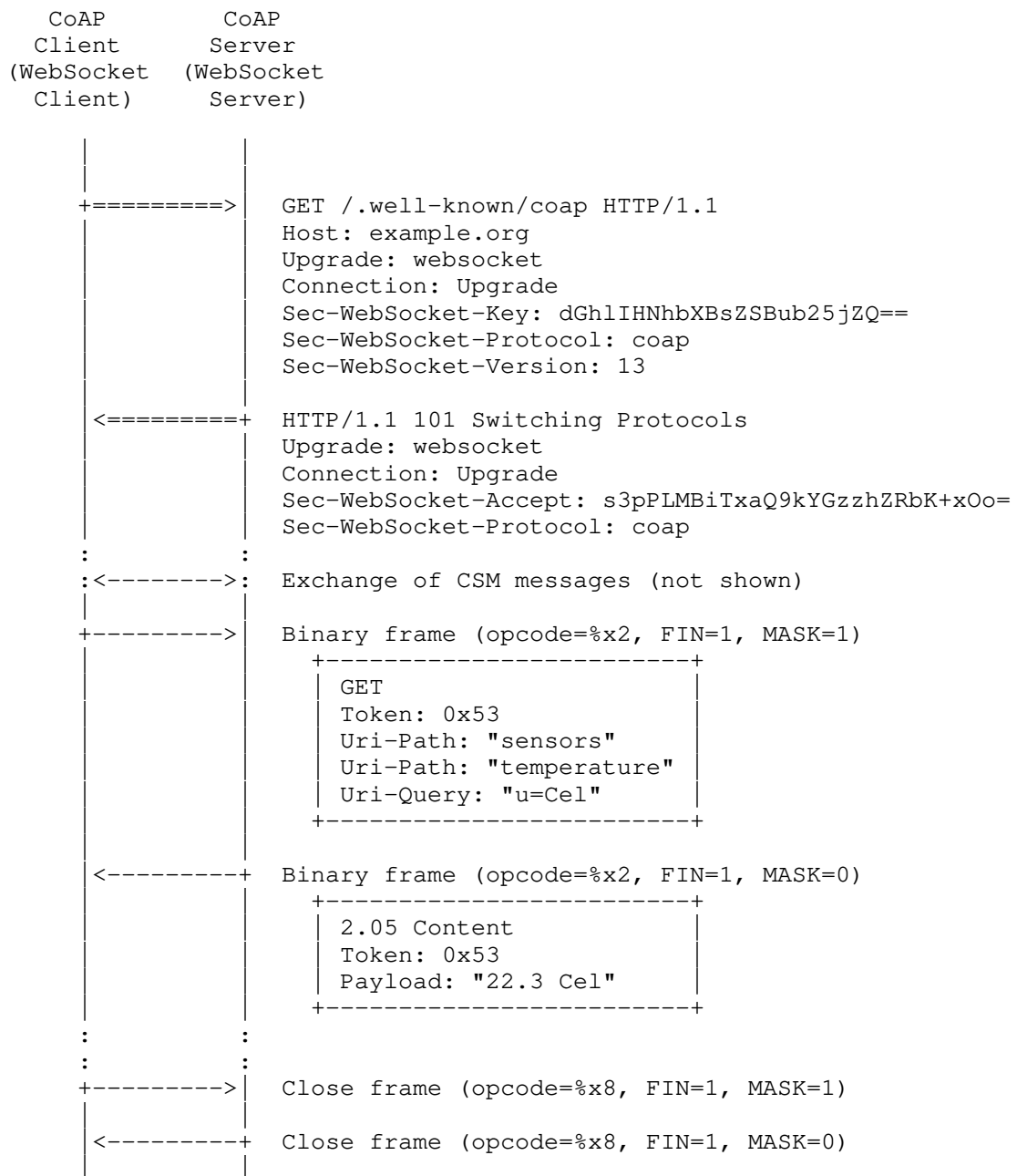


Figure 17: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

Figure 18 shows how a CoAP client uses a CoAP forward proxy with a WebSocket endpoint to retrieve the representation of the resource "coap://[2001:db8::1]/". The use of the forward proxy and the address of the WebSocket endpoint are determined by the client from local configuration rules. The request URI is specified in the Proxy-Uri Option. Since the request URI uses the "coap" URI scheme, the proxy fulfills the request by issuing a Confirmable GET request over UDP to the CoAP server and returning the response over the WebSocket connection to the client.

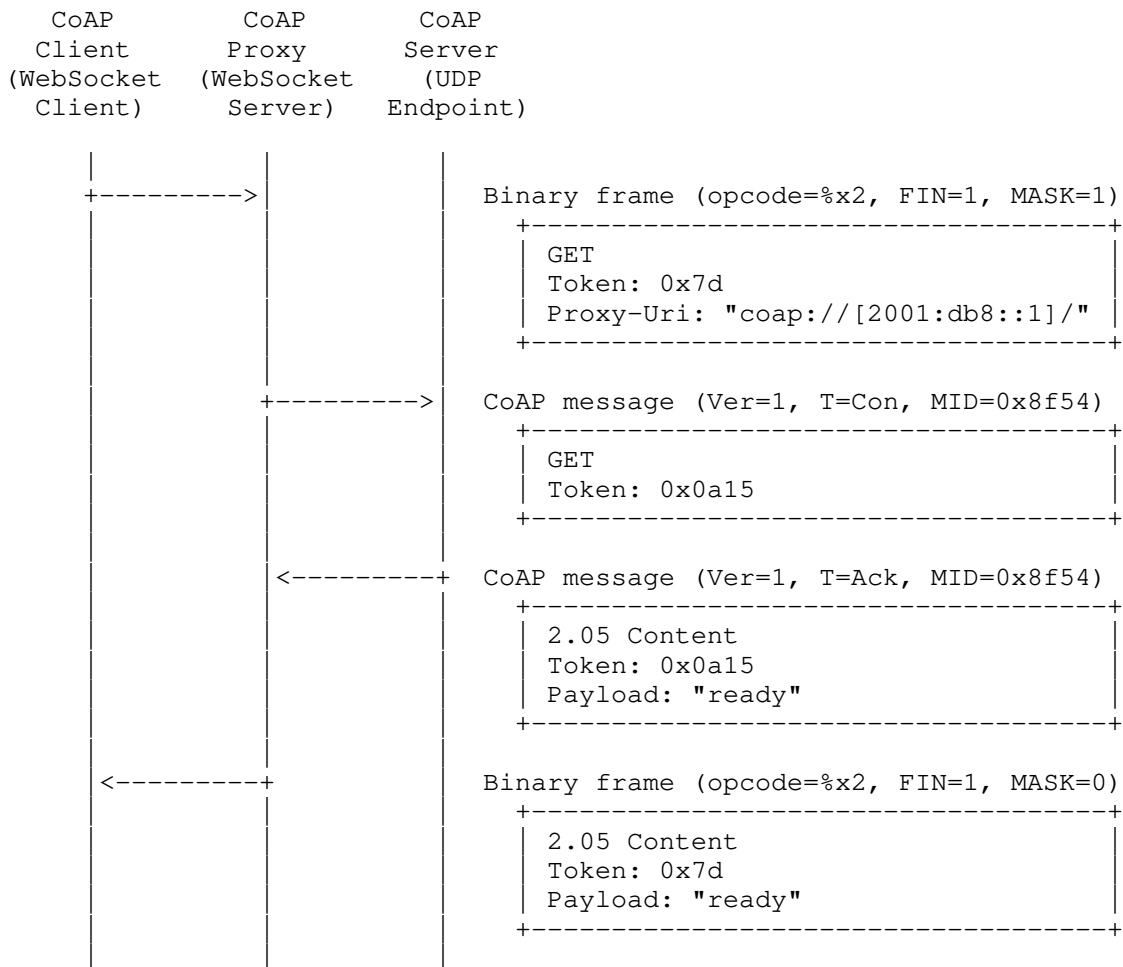


Figure 18: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSocket-enabled CoAP proxy

Appendix B. Change Log

The RFC Editor is requested to remove this section at publication.

B.1. Since draft-ietf-core-coap-tcp-tls-02

Merged draft-savolainen-core-coap-websockets-07 Merged draft-bormann-core-block-bert-01 Merged draft-bormann-core-coap-sig-02

B.2. Since draft-ietf-core-coap-tcp-tls-03

Editorial updates

Added mandatory exchange of Capabilities and Settings messages after connecting

Added support for coaps+tcp port 5684 and more details on Application-Layer Protocol Negotiation (ALPN)

Added guidance on CoAP Signaling Ping-Pong versus WebSocket Ping-Pong

Updated references and requirements for TLS security considerations

B.3. Since draft-ietf-core-coap-tcp-tls-04

Updated references

Added Appendix: Updates to RFC7641 Observing Resources in the Constrained Application Protocol (CoAP)

Updated Capability and Settings Message (CSM) exchange in the Opening Handshake to allow initiator to send messages before receiving acceptor CSM

B.4. Since draft-ietf-core-coap-tcp-tls-05

Addressed feedback from Working Group Last Call

Added Securing CoAP section and informative reference to OSCOAP

Removed the Server-Name and Bad-Server-Name Options

Clarified the Capability and Settings Message (CSM) exchange

Updated Pong response requirements

Added Connection Initiator and Connection Acceptor terminology where appropriate

Updated LWM2M 1.0 informative reference

B.5. Since draft-ietf-core-coap-tcp-tls-06

Addressed feedback from second Working Group Last Call

B.6. Since draft-ietf-core-coap-tcp-tls-07

Addressed feedback from IETF Last Call

Addressed feedback from ARTART review

Addressed feedback from GENART review

Addressed feedback from TSVART review

Added fragment identifiers to URI schemes

Added "Updates RFC7959" for BERT

Added "Updates RFC6455" to extend well-known URI mechanism to ws and wss

Clarified well-known URI mechanism use for all URI schemes

Changed NoSec to optional-to-implement

Acknowledgements

We would like to thank Stephen Berard, Geoffrey Cristallo, Olivier Delaby, Esko Dijk, Christian Groves, Nadir Javed, Michael Koster, Matthias Kovatsch, Achim Kraus, David Navarro, Szymon Sasin, Goran Selander, Zach Shelby, Andrew Summers, Julien Vermillard, and Gengyu Wei for their feedback.

Last-call reviews from Yoshifumi Nishida, Mark Nottingham, and Meral Shirazipour as well as several IESG reviewers provided extensive comments; from the IESG, we would like to specifically call out Ben Campbell, Mirja Kuehlewind, Eric Rescorla, Adam Roach, and the responsible AD Alexey Melnikov.

Contributors

Matthias Kovatsch
Siemens AG
Otto-Hahn-Ring 6
Munich D-81739

Phone: +49-173-5288856
EMail: matthias.kovatsch@siemens.com

Teemu Savolainen
Nokia Technologies
Hatanpaan valtatie 30
Tampere FI-33100
Finland

Email: teemu.savolainen@nokia.com

Valik Solorzano Barboza
Zebra Technologies
820 W. Jackson Blvd. Suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: vsolorzanobarboza@zebra.com

Authors' Addresses

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

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

Simon Lemay
Zebra Technologies
820 W. Jackson Blvd. Suite 700
Chicago 60607
United States of America

Phone: +1-847-634-6700
Email: slemay@zebra.com

Hannes Tschofenig
ARM Ltd.
110 Fulbourn Rd
Cambridge CB1 9NJ
Great Britain

Email: Hannes.tschofenig@gmx.net
URI: <http://www.tschofenig.priv.at>

Klaus Hartke
Universitaet Bremen TZI
Postfach 330440
Bremen D-28359
Germany

Phone: +49-421-218-63905
Email: hartke@tzi.org

Bilhanan Silverajan
Tampere University of Technology
Korkeakoulunkatu 10
Tampere FI-33720
Finland

Email: bilhanan.silverajan@tut.fi

Brian Raymor (editor)

Email: brianraymor@hotmail.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 $1 \times \text{RTO}$ and $\text{ACK_RANDOM_FACTOR} \times \text{RTO}$.

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: July 21, 2021

M. Veillette, Ed.
Trilliant Networks Inc.
P. van der Stok, Ed.
consultant
A. Pelov
Acklio
A. Bierman
YumaWorks
I. Petrov, Ed.
Acklio
January 17, 2021

CoAP Management Interface (CORECONF)
draft-ietf-core-comi-11

Abstract

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

Note

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

Status of This Memo

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

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

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

This Internet-Draft will expire on July 21, 2021.

Copyright Notice

Copyright (c) 2021 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. CORECONF Architecture	5
2.1. Major differences between RESTCONF and CORECONF	6
2.1.1. Differences due to CoAP and its efficient usage	6
2.1.2. Differences due to the use of CBOR	7
2.2. Compression of YANG identifiers	7
2.3. Instance-identifier	8
2.4. Media-Types	8
2.5. Unified datastore	10
3. Example syntax	11
4. CoAP Interface	11
4.1. Using the 'k' query parameter	13
4.2. Data Retrieval	15
4.2.1. Using the 'c' query parameter	15
4.2.2. Using the 'd' query parameter	16
4.2.3. GET	16
4.2.4. FETCH	19
4.3. Data Editing	20
4.3.1. Data Ordering	20
4.3.2. POST	20
4.3.3. PUT	21
4.3.4. iPATCH	22
4.3.5. DELETE	23
4.4. Full datastore access	24
4.4.1. Full datastore examples	24
4.5. Event stream	25
4.5.1. Notify Examples	26
4.5.2. The 'f' query parameter	27

4.6.	RPC statements	27
4.6.1.	RPC Example	28
5.	Use of Block-wise Transfers	30
6.	Application Discovery	30
6.1.	YANG library	30
6.2.	Resource Discovery	31
6.2.1.	Datastore Resource Discovery	31
6.2.2.	Data node Resource Discovery	32
6.2.3.	Event stream Resource Discovery	32
7.	Error Handling	33
8.	Security Considerations	36
9.	IANA Considerations	37
9.1.	Resource Type (rt=) Link Target Attribute Values Registry	37
9.2.	CoAP Content-Formats Registry	37
9.3.	Media Types Registry	38
9.4.	YANG Namespace Registration	39
10.	Acknowledgments	39
11.	References	40
11.1.	Normative References	40
11.2.	Informative References	41
Appendix A.	ietf-coreconf YANG module	42
Appendix B.	ietf-coreconf .sid file	48
Authors' Addresses		51

1. Introduction

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

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

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

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

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

1.1. Terminology

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

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

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

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

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

The following terms are defined in this document:

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

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

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

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

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

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

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

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

instance-value: The value assigned to a data node instance. Instance-values are serialized into the payload according to the rules defined in section 4 of [I-D.ietf-core-yang-cbor].

2. CORECONF Architecture

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

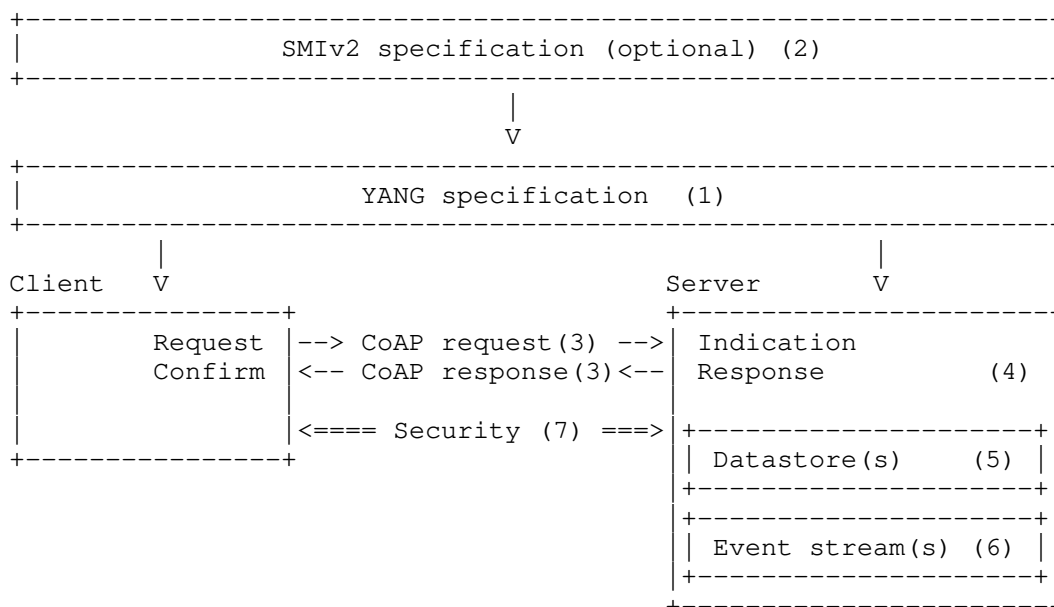


Figure 1: Abstract CORECONF architecture

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

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

- (2) SMIV2 specification: Optional part that consists of a named module which, specifies a set of variables and "conceptual tables". There is an algorithm to translate SMIV2 specifications to YANG specifications.
- (3) CoAP request/response messages: The CORECONF client sends request messages to and receives response messages from the CORECONF server.
- (4) Request, Indication, Response, Confirm: Processes performed by the CORECONF clients and servers.
- (5) Datastore: A resource used to access configuration data, state data, RPCs, and actions. A CORECONF server may support a single unified datastore or multiple datastores as those defined by Network Management Datastore Architecture (NMDA) [RFC8342].
- (6) Event stream: A resource used to get real-time notifications. A CORECONF server may support multiple Event streams serving different purposes such as normal monitoring, diagnostic, syslog, security monitoring.
- (7) Security: The server MUST prevent unauthorized users from reading or writing any CORECONF resources. CORECONF relies on security protocols such as DTLS [RFC6347] or OSCORE [RFC8613] to secure CoAP communications.

2.1. Major differences between RESTCONF and CORECONF

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

2.1.1. Differences due to CoAP and its efficient usage

- o CORECONF uses CoAP/UDP as transport protocol and CBOR as payload format [I-D.ietf-core-yang-cbor]. RESTCONF uses HTTP/TCP as transport protocol and JSON or XML as payload formats.
- o CORECONF uses the methods FETCH and iPATCH to access multiple data nodes. RESTCONF uses instead the HTTP method PATCH and the HTTP method GET with the "fields" Query parameter.
- o RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not supported by CoAP.

- o CORECONF does not support "insert" query parameter (first, last, before, after) and the "point" query parameter which are supported by RESTCONF.
- o CORECONF does not support the "start-time" and "stop-time" query parameters to retrieve past notifications.

2.1.2. Differences due to the use of CBOR

- o CORECONF encodes YANG identifier strings as numbers, where RESTCONF does not.
- o CORECONF also differ in the handling of default values, only 'report-all' and 'trim' options are supported.

2.2. Compression of YANG identifiers

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

When used in a URI, SIDs are encoded using base64 encoding of the SID bytes. The base64 encoding is using the URL and Filename safe alphabet as defined by [RFC4648] section 5, without padding. The last 6 bits encoded is always aligned with the least significant 6 bits of the SID represented using an unsigned integer. 'A' characters (value 0) at the start of the resulting string are removed. See Figure 2 for complete illustration.

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

Figure 2

For example, SID 1721 is encoded as follow.

```

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

```

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

2.3. Instance-identifier

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

When part of a payload, instance-identifiers are encoded in CBOR based on the rules defined in [I-D.ietf-core-yang-cbor] section 6.13.1. When part of a URI, the SID is appended to the URI of the targeted datastore, the keys are specified using the 'k' query parameter as defined in Section 4.1.

2.4. Media-Types

CORECONF uses Media-Types based on the YANG to CBOR mapping specified in [I-D.ietf-core-yang-cbor].

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

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

The following new Media-Types are defined in this document:

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

FORMAT: CBOR array of instance-identifier

The message payload of Media-Type 'application/yang-identifiers+cbor' is encoded using a CBOR array. Each entry of

this CBOR array contain an instance-identifier encoded as defined in [I-D.ietf-core-yang-cbor] section 6.13.1.

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

FORMAT: CBOR array of CBOR map of instance-identifier, instance-value

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

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

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

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

2.5. Unified datastore

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

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

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

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

3. Example syntax

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

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

4. CoAP Interface

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

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

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

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

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

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

The methods used by CORECONF are:

Operation	Description
GET	Retrieve the datastore resource or a data node resource
FETCH	Retrieve specific data nodes within a datastore resource
POST	Create a datastore resource or a data node resource, invoke an RPC or action
PUT	Create or replace a datastore resource or a data node resource
iPATCH	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 at most one instance of the 'k' query parameter for YANG list element selection for the GET, PUT, POST, and DELETE methods. Having multiple instances of that query parameter shall be treated as an error.

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

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

4.1. Using the 'k' query parameter

The 'k' (key) parameter specifies a specific instance of a data node. The SID in the URI is followed by the (?k=key1,key2,...). Where SID identifies a data node, and key1, key2 are the values of the key leaves that specify an instance. Lists can have multiple keys, and lists can be part of lists. The order of key value generation is given recursively by:

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

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

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

In this table:

- o The method int2str() is used to convert an integer value to a decimal string. For example, int2str(0x0123) return the three-character string "291".
- o The boolean values false and true are represented as the single-character strings "0" and "1" respectively.
- o The method urlSafeBase64() is used to convert a binary string to base64 using the URL and Filename safe alphabet as defined by [RFC4648] section 5, without padding. For example, urlSafeBase64(0xF956A13C) return the six-character string "-VahPA".
- o The method CBORencode() is used to convert a YANG value to CBOR as specified in [I-D.ietf-core-yang-cbor] section 6.

The resulting key strings are joined using commas between every two consecutive key values to produce the value of the 'k' parameter.

4.2. Data Retrieval

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

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

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

4.2.1. Using the 'c' query parameter

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

The allowed values are:

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

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

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

4.2.2. Using the 'd' query parameter

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

The allowed values are:

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

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

4.2.3. GET

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

FORMAT:

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

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

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

4.2.3.1. GET Examples

Using, for example, the current-datetime leaf from module ietf-system [RFC7317], a request is sent to retrieve the value of 'system-state/clock/current-datetime'. The SID of 'system-state/clock/current-datetime' is 1723, encoded in base64 according to Section 2.2, yields a7. The response to the request returns the CBOR map with the key set to the SID of the requested data node (i.e. 1723) and the value encoded using a 'text string' as defined in [I-D.ietf-core-yang-cbor] section 6.4. The datastore resource path /c is an example location discovered with a request similar to Figure 4.

REQ: GET </c/a7>

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

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

REQ: GET </c/a5>

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

Figure 3

This example shows the retrieval of the /interfaces/interface YANG list accessed using SID 1533 (base64: X9). The return payload is encoded using a CBOR array as specified by [I-D.ietf-core-yang-cbor] section 4.4.1 containing 2 instances.

REQ: GET </c/X9>

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

To retrieve a specific instance within the /interfaces/interface YANG list, the CORECONF client adds the key of the targeted instance in its CoAP request using the 'k' query parameter. The return payload containing the instance requested is encoded using a CBOR array as specified by [I-D.ietf-core-yang-cbor] section 4.4.1 containing the requested instance.

REQ: GET </c/X9?k=eth0>

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

It is equally possible to select a leaf of a specific instance of a list. The example below requests the description leaf (SID 1534, base64: X-) within the interface list corresponding to the interface

name "eth0". The returned value is encoded in CBOR based on the rules specified by [I-D.ietf-core-yang-cbor] section 6.4.

REQ: GET </c/X-?k=eth0>

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

4.2.4. FETCH

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

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

For compactness, indexes of the list instance identifiers returned by the FETCH response SHOULD be elided, only the SID is provided. This approach may also help reducing implementations complexity since the format of each entry within the CBOR array of the FETCH response is identical to the format of the corresponding GET response.

FORMAT:

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

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

4.2.4.1. FETCH examples

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

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

RES: 2.05 Content (Content-Format: application/yang-instances+cbor)
[
  {
    1723 : "2014-10-26T12:16:31Z" / current-datetime (SID 1723) /
  },
  {
    1533 : {
      4 : "eth0",          / name (SID 1537) /
      1 : "Ethernet adaptor", / description (SID 1534) /
      5 : 1880,            / type (SID 1538), identity /
                          / ethernetCsmacd (SID 1880) /
      2 : true,            / enabled (SID 1535) /
      11 : 3               / oper-status (SID 1544), value is testing /
    }
  }
]
```

4.3. Data Editing

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

4.3.1. Data Ordering

A CORECONF server MUST preserve the relative order of all user-ordered list and leaf-list entries that are received in a single edit request. These YANG data node types are encoded as CBOR arrays so messages will preserve their order.

4.3.2. POST

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

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

FORMAT:

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

2.01 Created

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

4.3.2.1. Post example

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

```
REQ: POST <coap://example.com/ds/X9>
(Content-Format: application/yang-data+cbor; id=sid)
```

```
{
  1533 : [
    {
      4 : "eth5",           / name (SID 1537) /
      1 : "Ethernet adaptor", / description (SID 1534) /
      5 : 1880,             / type (SID 1538), identity /
                           / ethernetCsmacd (SID 1880) /
      2 : true              / enabled (SID 1535) /
    }
  ]
}
```

```
RES: 2.01 Created
```

4.3.3. PUT

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

FORMAT:

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

2.01 Created

4.3.3.1. PUT example

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

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

RES: 2.04 Changed

4.3.4. iPATCH

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

There are no query parameters for the iPATCH method.

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

FORMAT:

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

2.04 Changed

4.3.4.1. iPATCH example

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

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

REQ: iPATCH </c>

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

```
[
  {
    1755 : true                / enabled (SID 1755) /
  },
  {
    [1756, "tac.nrc.ca"] : null / server (SID 1756) /
  },
  {
    1756 : {
      3 : "tic.nrc.ca",        / name (SID 1759) /
      4 : true,                / prefer (SID 1760) /
      5 : {
        1 : "132.246.11.231"   / address (SID 1762) /
      }
    }
  }
]
```

RES: 2.04 Changed

4.3.5. DELETE

A data node resource is deleted with the DELETE method.

FORMAT:

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

2.02 Deleted

4.3.5.1. DELETE example

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

REQ: DELETE </c/X9?k=eth0>

RES: 2.02 Deleted

4.4. Full datastore access

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

FORMAT:

GET <datastore resource>

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

FORMAT:

PUT <datastore resource>

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

2.04 Changed

FORMAT:

POST <datastore resource>

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

2.01 Created

FORMAT:

DELETE <datastore resource>

2.02 Deleted

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

4.4.1. Full datastore examples

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

REQ: GET </c>

RES: 2.05 Content (Content-Format: application/yang-data+cbor; id=sid)

```
{
  1721 : {
    2 : "2016-10-26T12:16:31Z", / current-datetime (SID 1723) /
    1 : "2014-10-05T09:00:00Z" / boot-datetime (SID 1722) /
  },
  1533 : [
    {
      4 : "eth0", / name (SID 1537) /
      1 : "Ethernet adaptor", / description (SID 1534) /
      5 : 1880, / type (SID 1538), identity: /
      / ethernetCsmacd (SID 1880) /
      2 : true, / enabled (SID 1535) /
      11 : 3 / oper-status (SID 1544), value is testing /
    }
  ]
}
```

4.5. Event stream

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

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

Each response payload carries one or multiple notifications. The number of notifications reported, and the conditions used to remove notifications from the reported list are left to implementers. When multiple notifications are reported, they MUST be ordered starting from the newest notification at index zero. Note that this could lead to notifications being sent multiple times, which increases the probability for the client to receive them, but it might potentially lead to messages that exceed the MTU of a single CoAP packet. If such cases could arise, implementers should make sure appropriate fragmentation is available - for example the one described in Section 5.

The format of notification without any content is a null value. The format of single notification is defined in [I-D.ietf-core-yang-cbor]

section 4.2.1. For multiple notifications the format is an array where each element is a single notification as described in [I-D.ietf-core-yang-cbor] section 4.2.1.

FORMAT:

GET <stream-resource> Observe(0)

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

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

An example implementation is:

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

4.5.1. Notify Examples

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

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

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


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

RES:  2.05 Content (Content-Format: application/yang-tree+cbor)
      Observe(12)
[
  {
    60010 : {
      1 : "0/4/21",      / port-name (SID 60011) /
      2 : "Open pin 2"   / port-fault (SID 60012) /
    }
  },
  {
    60010 : {
      1 : "1/4/21",      / port-name (SID 60011) /
      2 : "Open pin 5"   / port-fault (SID 60012) /
    }
  }
]
```

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

4.5.2. The 'f' query parameter

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

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

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

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

4.6. RPC statements

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

The request payload contains the values assigned to the input container when specified. The response payload contains the values of the output container when specified. Both the input and output

containers are encoded in CBOR using the rules defined in [I-D.ietf-core-yang-cbor] section 4.2.1.

The returned success response code is 2.05 Content.

FORMAT:

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

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

4.6.1. RPC Example

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

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

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

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

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

REQ: POST </c/Opq?k=myserver>

(Content-Format: application/yang-data+cbor; id=sid)

```
{
  60002 : {
    1 : "2016-02-08T14:10:08Z09:00" / reset-at (SID 60003) /
  }
}
```

RES: 2.05 Content (Content-Format: application/yang-data+cbor; id=sid)

```
{
  60002 : {
    2 : "2016-02-08T14:10:08Z09:18" / reset-finished-at (SID 60004)/
  }
}
```

5. Use of Block-wise Transfers

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

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

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

6. Application Discovery

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

6.1. YANG library

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

As described in [RFC6690], the location of the YANG library can be found by sending a GET request to `"/.well-known/core"` including a

resource type (RT) parameter with the value "core.c.yml". Upon success, the return payload will contain the root resource of the YANG library module.

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

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

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

6.2. Resource Discovery

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

6.2.1. Datastore Resource Discovery

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

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

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

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

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

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

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

Figure 4

6.2.2. Data node Resource Discovery

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

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

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

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

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

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

6.2.3. Event stream Resource Discovery

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

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

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

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

Figure 5

7. Error Handling

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

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

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

- o Error 4.01 (Unauthorized) is returned by the CORECONF server when the CORECONF client is not authorized to perform the requested action on the targeted resource (i.e. data node, datastore, rpc, action or event stream).
- o Error 4.02 (Bad Option) is returned by the CORECONF server when one or more CoAP options are unknown or malformed.
- o Error 4.04 (Not Found) is returned by the CORECONF server when the CORECONF client is requesting a non-instantiated resource (i.e. data node, datastore, rpc, action or event stream).
- o Error 4.05 (Method Not Allowed) is returned by the CORECONF server when the CORECONF client is requesting a method not supported on the targeted resource. (e.g. GET on an rpc, PUT or POST on a data node with "config" set to false).
- o Error 4.08 (Request Entity Incomplete) is returned by the CORECONF server if one or multiple blocks of a block transfer request is missing, see [RFC7959] for more details.
- o Error 4.13 (Request Entity Too Large) may be returned by the CORECONF server during a block transfer request, see [RFC7959] for more details.

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

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

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

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

- o error-tag 'operation-failed' is returned by the CORECONF server when the operation request cannot be processed successfully.
 - * error-app-tag 'malformed-message' is returned by the CORECONF server when the payload received from the CORECONF client does not contain a well-formed CBOR content as defined in [RFC7049] section 3.3 or does not comply with the CBOR structure defined within this document.
 - * error-app-tag 'data-not-unique' is returned by the CORECONF server when the validation of the 'unique' constraint of a list or leaf-list fails.
 - * error-app-tag 'too-many-elements' is returned by the CORECONF server when the validation of the 'max-elements' constraint of a list or leaf-list fails.
 - * error-app-tag 'too-few-elements' is returned by the CORECONF server when the validation of the 'min-elements' constraint of a list or leaf-list fails.
 - * error-app-tag 'must-violation' is returned by the CORECONF server when the restrictions imposed by a 'must' statement are violated.

- * error-app-tag 'duplicate' is returned by the CORECONF server when a client tries to create a duplicate list or leaf-list entry.
- o error-tag 'invalid-value' is returned by the CORECONF server when the CORECONF client tries to update or create a leaf with a value encoded using an invalid CBOR datatype or if the 'range', 'length', 'pattern' or 'require-instance' constrain is not fulfilled.
 - * error-app-tag 'invalid-datatype' is returned by the CORECONF server when CBOR encoding does not follow the rules set by the YANG Build-In type or when the value is incompatible with it (e.g. a value greater than 127 for an int8, undefined enumeration).
 - * error-app-tag 'not-in-range' is returned by the CORECONF server when the validation of the 'range' property fails.
 - * error-app-tag 'invalid-length' is returned by the CORECONF server when the validation of the 'length' property fails.
 - * error-app-tag 'pattern-test-failed' is returned by the CORECONF server when the validation of the 'pattern' property fails.
- o error-tag 'missing-element' is returned by the CORECONF server when the operation requested by a CORECONF client fails to comply with the 'mandatory' constraint defined. The 'mandatory' constraint is enforced for leafs and choices, unless the node or any of its ancestors have a 'when' condition or 'if-feature' expression that evaluates to 'false'.
 - * error-app-tag 'missing-key' is returned by the CORECONF server to further qualify a missing-element error. This error is returned when the CORECONF client tries to create or list instance, without all the 'key' specified or when the CORECONF client tries to delete a leaf listed as a 'key'.
 - * error-app-tag 'missing-input-parameter' is returned by the CORECONF server when the input parameters of an RPC or action are incomplete.
- o error-tag 'unknown-element' is returned by the CORECONF server when the CORECONF client tries to access a data node of a YANG module not supported, of a data node associated with an 'if-feature' expression evaluated to 'false' or to a 'when' condition evaluated to 'false'.

- o error-tag 'bad-element' is returned by the CORECONF server when the CORECONF client tries to create data nodes for more than one case in a choice.
- o error-tag 'data-missing' is returned by the CORECONF server when a data node required to accept the request is not present.
- * error-app-tag 'instance-required' is returned by the CORECONF server when a leaf of type 'instance-identifier' or 'leafref' marked with require-instance set to 'true' refers to an instance that does not exist.
- * error-app-tag 'missing-choice' is returned by the CORECONF server when no nodes exist in a mandatory choice.
- o error-tag 'error' is returned by the CORECONF server when an unspecified error has occurred.

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

```
RES: 4.00 Bad Request (Content-Format: application/yang-data+cbor; id=sid)
{
  1024 : {
    4 : 1011,          / error-tag (SID 1028) /
                      /   = invalid-value (SID 1011) /
    1 : 1018,          / error-app-tag (SID 1025) /
                      /   = not-in-range (SID 1018) /
    2 : 1740,          / error-data-node (SID 1026) /
                      /   = timezone-utc-offset (SID 1740) /
    3 : "maximum value exceeded" / error-message (SID 1027) /
  }
}
```

8. Security Considerations

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

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

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

As [I-D.ietf-core-yang-cbor] and [RFC4648] are used for payload and SID encoding, the security considerations of those documents also need to be well-understood.

9. IANA Considerations

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

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

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

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

9.2. CoAP Content-Formats Registry

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

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

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

9.3. Media Types Registry

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

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

Each of these media types share the following information:

- o Subtype name: <as listed in table>
- o Required parameters: N/A
- o Optional parameters: N/A
- o Encoding considerations: binary
- o Security considerations: See the Security Considerations section of RFC XXXX
- o Interoperability considerations: N/A
- o Published specification: RFC XXXX
- o Applications that use this media type: CORECONF
- 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.

9.4. YANG Namespace Registration

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

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

Registrant Contact: The IESG.

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

Reference: RFC XXXX

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

10. Acknowledgments

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

Mehmet Ersue and Bert Wijnen explained the encoding aspects of PDUs transported under SNMP. 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, Klaus Hartke, Michael van Hartskamp, Tanguy Ropitault, Juergen Schoenwaelder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michael Verschoor, and Thomas Watteyne.

11. References

11.1. Normative References

- [I-D.ietf-core-sid]
Veillette, M., Pelov, A., and I. Petrov, "YANG Schema Item Identifier (YANG SID)", draft-ietf-core-sid-14 (work in progress), July 2020.
- [I-D.ietf-core-yang-cbor]
Veillette, M., Petrov, I., and A. Pelov, "CBOR Encoding of Data Modeled with YANG", draft-ietf-core-yang-cbor-13 (work in progress), July 2020.
- [I-D.ietf-core-yang-library]
Veillette, M. and I. Petrov, "Constrained YANG Module Library", draft-ietf-core-yang-library-03 (work in progress), January 2021.
- [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>>.
- [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>>.
- [RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <<https://www.rfc-editor.org/info/rfc5277>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <<https://www.rfc-editor.org/info/rfc6243>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <<https://www.rfc-editor.org/info/rfc8040>>.
- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

11.2. Informative References

- [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-36 (work in progress), November 2020.
- [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>>.

- [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>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

Appendix A. ietf-coreconf YANG module

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

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

  import ietf-datastores {
    prefix ds;
  }

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

  organization
    "IETF Core Working Group";

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

     Alexander Pelov
     <mailto:alexander@ackl.io>
```


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

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

description

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

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

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

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

```
revision 2019-03-28 {
  description
    "Initial revision.";
  reference
    "[I-D.ietf-core-comi] CoAP Management Interface";
}

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

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

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

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

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

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

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

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

identity error {
  base error-tag;
  description
```

```
    "Returned by the CORECONF server when an unspecified error has
    occurred.";
}

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

identity malformed-message {
  base error-app-tag;
  description
    "Returned by the CORECONF server when the payload received
    from the CORECONF client don't contain a well-formed CBOR
    content as defined in [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 CORECONF server when the validation of the
    'unique' constraint of a list or leaf-list fails.";
}

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

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

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

identity duplicate {
```

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

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

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

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

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

identity missing-input-parameter {
    base error-app-tag;
```

```
    description
      "Returned by the CORECONF server when the input parameters
        of a RPC or action are incomplete.";
  }

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

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

  rc:yang-data coreconf-error {
    container error {
      description
        "Optional payload of a 4.00 Bad Request CoAP error.";

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

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

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

```
    }

    leaf error-message {
      type string;
      description
        "A message describing the error.";
    }
  }
}
<CODE ENDS>
```

Appendix B. ietf-coreconf .sid file

```
{
  "assignment-ranges": [
    {
      "entry-point": 1000,
      "size": 100
    }
  ],
  "module-name": "ietf-coreconf",
  "module-revision": "2019-03-28",
  "items": [
    {
      "namespace": "module",
      "identifier": "ietf-coreconf",
      "sid": 1000
    },
    {
      "namespace": "identity",
      "identifier": "bad-element",
      "sid": 1001
    },
    {
      "namespace": "identity",
      "identifier": "data-missing",
      "sid": 1002
    },
    {
      "namespace": "identity",
      "identifier": "data-not-unique",
      "sid": 1003
    },
    {
      "namespace": "identity",
      "identifier": "duplicate",
      "sid": 1004
    }
  ]
}
```

```
,
{
  "namespace": "identity",
  "identifier": "error",
  "sid": 1005
},
{
  "namespace": "identity",
  "identifier": "error-app-tag",
  "sid": 1006
},
{
  "namespace": "identity",
  "identifier": "error-tag",
  "sid": 1007
},
{
  "namespace": "identity",
  "identifier": "instance-required",
  "sid": 1008
},
{
  "namespace": "identity",
  "identifier": "invalid-datatype",
  "sid": 1009
},
{
  "namespace": "identity",
  "identifier": "invalid-length",
  "sid": 1010
},
{
  "namespace": "identity",
  "identifier": "invalid-value",
  "sid": 1011
},
{
  "namespace": "identity",
  "identifier": "malformed-message",
  "sid": 1012
},
{
  "namespace": "identity",
  "identifier": "missing-choice",
  "sid": 1013
},
{
  "namespace": "identity",
```

```
    "identifier": "missing-element",
    "sid": 1014
  },
  {
    "namespace": "identity",
    "identifier": "missing-input-parameter",
    "sid": 1015
  },
  {
    "namespace": "identity",
    "identifier": "missing-key",
    "sid": 1016
  },
  {
    "namespace": "identity",
    "identifier": "must-violation",
    "sid": 1017
  },
  {
    "namespace": "identity",
    "identifier": "not-in-range",
    "sid": 1018
  },
  {
    "namespace": "identity",
    "identifier": "operation-failed",
    "sid": 1019
  },
  {
    "namespace": "identity",
    "identifier": "pattern-test-failed",
    "sid": 1020
  },
  {
    "namespace": "identity",
    "identifier": "too-few-elements",
    "sid": 1021
  },
  {
    "namespace": "identity",
    "identifier": "too-many-elements",
    "sid": 1022
  },
  {
    "namespace": "identity",
    "identifier": "unified",
    "sid": 1029
  },
  },
```



```
{
  "namespace": "identity",
  "identifier": "unknown-element",
  "sid": 1023
},
{
  "namespace": "data",
  "identifier": "/ietf-coreconf:error",
  "sid": 1024
},
{
  "namespace": "data",
  "identifier": "/ietf-coreconf:error/error-app-tag",
  "sid": 1025
},
{
  "namespace": "data",
  "identifier": "/ietf-coreconf:error/error-data-node",
  "sid": 1026
},
{
  "namespace": "data",
  "identifier": "/ietf-coreconf:error/error-message",
  "sid": 1027
},
{
  "namespace": "data",
  "identifier": "/ietf-coreconf:error/error-tag",
  "sid": 1028
}
]
}
```

Authors' Addresses

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

Email: michel.veillette@trilliant.com

Peter van der Stok (editor)
consultant

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

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

Email: a@ackl.io

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

Email: andy@yumaworks.com

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

Email: ivaylo@ackl.io

CoRE Working Group
Internet-Draft
Intended status: Informational
Expires: January 13, 2022

M. Koster
SmartThings
B. Silverajan, Ed.
Tampere University
July 12, 2021

Dynamic Resource Linking for Constrained RESTful Environments
draft-ietf-core-dynlink-14

Abstract

This specification defines Link Bindings, which provide dynamic linking of state updates between resources, either on an endpoint or between endpoints, for systems using CoAP (RFC7252).

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 13, 2022.

Copyright Notice

Copyright (c) 2021 IETF Trust and the persons identified as the document authors. All rights reserved.

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

to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Terminology	3
3. Link Bindings	3
3.1. The "bind" attribute and Binding Methods	4
3.1.1. Polling	5
3.1.2. Observe	5
3.1.3. Push	5
3.1.4. Execute	6
3.2. Link Relation	6
4. Binding Table	6
5. Implementation Considerations	8
6. Security Considerations	8
7. IANA Considerations	8
7.1. Resource Type value 'core.bnd'	8
7.2. Link Relation Type	8
8. Acknowledgements	9
9. Contributors	9
10. Changelog	10
11. References	12
11.1. Normative References	12
11.2. Informative References	13
Authors' Addresses	13

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.

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], [RFC6690] and [RFC7641]. 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.

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 content updates between a source resource and a destination resource. The destination resource MAY be a group resource if the authority component of the destination URI contains a group address (either a multicast address or a name that resolves to a multicast address).

Since a binding is unidirectional, the binding entry defining a relationship is present only on one endpoint. The binding entry may be located either on the source or the destination endpoint depending on the binding method.

Conditional Notification Attributes defined in [I-D.ietf-core-conditional-attributes] can be used with Link Bindings in order to customize the notification behavior and timing.

3.1. The "bind" attribute and Binding Methods

A binding method defines the rules to generate the network-transfer exchanges that synchronize state between source and destination resources. By using REST methods content is sent from the source resource to the destination resource.

This specification defines a new CoRE link attribute "bind". This is the identifier for a binding method which defines the rules to synchronize the destination resource. This attribute is mandatory.

Attribute	Parameter	Value
Binding method	bind	xs: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
Execute	exec	Source	POST

Table 2: Binding Method Summary

The description of a binding method defines the following aspects:

Identifier: This is the value of the "bind" attribute used to identify the method.

Location: This information indicates whether the binding entry is stored on the source or on the destination endpoint.

REST Method: This is the REST method used in the Request/Response exchanges.

Conditional Notification: How Conditional Notification Attributes defined in [I-D.ietf-core-conditional-attributes] are used in the 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 defined in [I-D.ietf-core-conditional-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 parameters in the Observe request (see [I-D.ietf-core-conditional-attributes]).

3.1.3. Push

The Push method can be used to allow a source endpoint to replace an outdated resource state at the destination with a newer representation. When the Push method is assigned to a binding, the source endpoint sends PUT requests to the destination resource when the Conditional Notification Attributes are satisfied for the source resource. The source endpoint SHOULD only send a notification request if any included Conditional Notification Attributes are met.

The binding entry for this method MUST be stored on the source endpoint.

3.1.4. Execute

An alternative means for a source endpoint to deliver change-of-state notifications to a destination resource is to use the Execute Method. While the Push method simply updates the state of the destination resource with the representation of the source resource, Execute can be used when the destination endpoint wishes to receive all state changes from a source. This allows, for example, the existence of a resource collection consisting of all the state changes at the destination endpoint. When the Execute method is assigned to a binding, the source endpoint sends POST requests to the destination resource when the Conditional Notification Attributes are satisfied for the source resource. The source endpoint SHOULD only send a notification request if any included Conditional Notification Attributes are met. The binding entry for this method MUST be stored on the source endpoint.

Note: Both the Push and the Execute methods are examples of Server Push mechanisms that are being researched in the Thing-to-Thing Research Group (T2TRG) [I-D.irtf-t2trg-rest-iot].

3.2. Link Relation

Since Binding involves the creation of a link between two resources, Web Linking and the CoRE Link-Format used to represent binding information. This involves the creation of a new relation type, "boundto". In a Web link with this relation type, the target URI contains the location of the source resource and the context URI points to the destination resource.

4. Binding Table

The Binding Table is a special resource that describes the bindings on an endpoint. An endpoint offering a representation of the Binding Table resource SHOULD indicate its presence and enable its discovery by advertising a link at `"/.well-known/core"` [RFC6690]. If so, the Binding Table resource MUST be discoverable by using the Resource Type (rt) `'core.bnd'`.

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

Resource	rt=	Methods	Content-Format
Binding Table	core.bnd	GET, PUT	link-format

Table 3: Binding Table Description

The REST methods GET and PUT are used to manipulate a Binding Table. A GET request simply returns the current state of a Binding Table. A request with a PUT method and a content format of application/link-format is used to clear the bindings to the table or replaces its entire contents. All links in the payload of a PUT request MUST have a relation type "boundto".

The following example shows requests for discovering, retrieving and replacing bindings in a binding table.

```
Req: GET /.well-known/core?rt=core.bnd (application/link-format)
Res: 2.05 Content (application/link-format)
</bnd/>;rt=core.bnd;ct=40
```

```
Req: GET /bnd/
Res: 2.05 Content (application/link-format)
<coap://sensor.example.com/a/switch1/>;
    rel=boundto;anchor=/a/fan;bind="obs",
<coap://sensor.example.com/a/switch2/>;
    rel=boundto;anchor=/a/light;bind="obs"
```

```
Req: PUT /bnd/ (Content-Format: application/link-format)
<coap://sensor.example.com/s/light>;
    rel="boundto";anchor="/a/light";bind="obs";pmin=10;pmax=60
Res: 2.04 Changed
```

```
Req: GET /bnd/
Res: 2.05 Content (application/link-format)
<coap://sensor.example.com/s/light>;
    rel="boundto";anchor="/a/light";bind="obs";pmin=10;pmax=60
```

Figure 1: Binding Table Example

Additional operations on the Binding Table can be specified in future documents. Such operations can include, for example, the usage of the iPATCH or PATCH methods [RFC8132] for fine-grained addition and removal of individual bindings or binding subsets.

5. Implementation 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. Implementation considerations have to be given to how to monitor transactions made by the configuration tool with regards to Link Bindings, as well as any errors that may arise with establishing Link Bindings in addition to established Link Bindings.

6. Security Considerations

Consideration has to be given to what kinds of security credentials the state machine of a configuration tool or an embedded client needs to be configured with, and what kinds of access control lists client implementations should possess, so that transactions on creating Link Bindings and handling error conditions can be processed by the state machine.

7. IANA Considerations

7.1. Resource Type value 'core.bnd'

This specification registers a new Resource Type Link Target Attribute 'core.bnd' in the Resource Type (rt=) registry established as per [RFC6690].

Attribute Value: core.bnd

Description: See Section 4. This attribute value is used to discover the resource representing a binding table, which describes the link bindings between source and destination resources for the purposes of synchronizing their content.

Reference: This specification. Note to RFC editor: please insert the RFC of this specification.

Notes: None

7.2. 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

8. Acknowledgements

Acknowledgement is given to colleagues from the SENSEI project who were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further requirements for interface types have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this specification. Christian Amsuss supplied a comprehensive review of draft -06. Discussions with Ari Keraenen led to the addition of an extra binding method supporting POST operations.

9. Contributors

Christian Groves
Australia
email: cngroves.std@gmail.com

Zach Shelby
ARM
Vuokatti
FINLAND
phone: +358 40 7796297
email: zach.shelby@arm.com

Matthieu Vial
Schneider-Electric
Grenoble
France
phone: +33 (0)47657 6522
eMail: matthieu.vial@schneider-electric.com

Jintao Zhu
Huawei
Xi'an, Shaanxi Province
China
email: jintao.zhu@huawei.com

10. Changelog

draft-ietf-core-dynlink-14

- o Conditional Attributes section removed and submitted as draft-ietf-core-conditional-attributes-00

draft-ietf-core-dynlink-13

- o Conditional Attributes section restructured
- o "edge" and "con" attributes added
- o Implementation considerations, clarifications added when pmax == pmin
- o rewritten to remove talk of server reporting values to clients

draft-ietf-core-dynlink-12

- o Attributes epmin and epmax included
- o pmax now can be equal to pmin

draft-ietf-core-dynlink-11

- o Updates to author list

draft-ietf-core-dynlink-10

- o Binding methods now support both POST and PUT operations for server push.

draft-ietf-core-dynlink-09

- o Corrections in Table 1, Table 2, Figure 2.
- o Clarifications for additional operations to binding table added in section 5
- o Additional examples in Appendix A

draft-ietf-core-dynlink-08

- o Reorganize the draft to introduce Conditional Notification Attributes at the beginning

- o Made pmin and pmax type xs:decimal to accommodate fractional second timing
- o updated the attribute descriptions. lt and gt notify on all crossings, both directions
- o updated Binding Table description, removed interface description but introduced core.bnd rt attribute value

draft-ietf-core-dynlink-07

- o Added reference code to illustrate attribute interactions for observations

draft-ietf-core-dynlink-06

- o Document restructure and refactoring into three main sections
- o Clarifications on band usage
- o Implementation considerations introduced
- o Additional text on security considerations

draft-ietf-core-dynlink-05

- o Addition of a band modifier for gt and lt, adapted from draft-groves-core-obsattr
- o Removed statement prescribing gt MUST be greater than lt

draft-ietf-core-dynlink-03

- o General: Reverted to using "gt" and "lt" from "gth" and "lth" for this draft owing to concerns raised that the attributes are already used in LwM2M with the original names "gt" and "lt".
- o New author and editor added.

draft-ietf-core-dynlink-02

- o General: Changed the name of the greater than attribute "gt" to "gth" and the name of the less than attribute "lt" to "lth" due to conflict with the core resource directory draft lifetime "lt" attribute.
- o Clause 6.1: Addressed the editor's note by changing the link target attribute to "core.binding".

- o Added Appendix A for examples.

draft-ietf-core-dynlink-01

- o General: The term state synchronization has been introduced to describe the process of synchronization between destination and source resources.
- o General: The document has been restructured to make the information flow better.
- o Clause 3.1: The descriptions of the binding attributes have been updated to clarify their usage.
- o Clause 3.1: A new clause has been added to discuss the interactions between the resources.
- o Clause 3.4: Has been simplified to refer to the descriptions in 3.1. As the text was largely duplicated.
- o Clause 4.1: Added a clarification that individual resources may be removed from the binding table.
- o Clause 6: 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.

11. References

11.1. Normative References

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

- [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>>.

11.2. Informative References

- [I-D.ietf-core-conditional-attributes]
Koster, M. and B. Silverajan, "Conditional Attributes for Constrained RESTful Environments", draft-ietf-core-conditional-attributes-00 (work in progress), July 2021.
- [I-D.irtf-t2trg-rest-iot]
Keranen, A., Kovatsch, M., and K. Hartke, "RESTful Design for Internet of Things Systems", draft-irtf-t2trg-rest-iot-07 (work in progress), February 2021.
- [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>>.
- [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>>.

Authors' Addresses

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Email: michael.koster@smarththings.com

Bilhanan Silverajan (editor)
Tampere University
Kalevantie 4
Tampere FI-33100
Finland

Email: bilhanan.silverajan@tuni.fi

core
Internet-Draft
Intended status: Standards Track
Expires: May 18, 2017

P. van der Stok
Consultant
C. Bormann
Universitaet Bremen TZI
A. Sehgal
Consultant
November 14, 2016

Patch and Fetch Methods for Constrained Application Protocol (CoAP)
draft-ietf-core-etch-04

Abstract

The methods defined in RFC 7252 for the Constrained Application Protocol (CoAP) only allow access to a complete resource, not to parts of a resource. In case of resources with larger or complex data, or in situations where a resource continuity is required, replacing or requesting the whole resource is undesirable. Several applications using CoAP will need to perform partial resource accesses.

This specification defines the new CoAP methods, FETCH, PATCH and iPATCH, which are used to access and update parts of a resource.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 18, 2017.

Copyright Notice

Copyright (c) 2016 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
1.1. FETCH	3
1.2. PATCH and iPATCH	4
1.3. Requirements Language	4
1.4. Terminology and Acronyms	5
2. FETCH Method	5
2.1. Response Codes	6
2.2. Error Handling	6
2.3. Option Numbers	7
2.3.1. The Content-Format Option	7
2.3.2. The ETag Option	7
2.4. Working with Observe	7
2.5. Working with Block	8
2.6. Building FETCH Requests	8
2.7. A Simple Example for FETCH	8
3. PATCH and iPATCH Methods	9
3.1. Simple Examples for PATCH and iPATCH	11
3.2. Response Codes	13
3.3. Option Numbers	13
3.4. Error Handling	13
4. The New Set of CoAP Methods	15
5. Security Considerations	16
6. IANA Considerations	16
7. Change log	17
8. References	18
8.1. Normative References	18
8.2. Informative References	18
Acknowledgements	19
Authors' Addresses	19

1. Introduction

Similar to HTTP, the GET method defined in [RFC7252] for the Constrained Application Protocol (CoAP) only allows the specification of a URI and request parameters in CoAP options, not the transfer of a request payload detailing the request. This leads to some

applications to using POST where actually a cacheable, idempotent, safe request is desired.

Again similar to the original specification of HTTP, the PUT method defined in [RFC7252] only allows to replace a complete resource. This also leads applications to use POST where actually a cacheable, possibly idempotent request is desired.

The present specification adds new CoAP methods: FETCH, to perform the equivalent of a GET with a request body; and the twin methods PATCH and iPATCH, to modify parts of a CoAP resource.

1.1. FETCH

The CoAP GET method [RFC7252] is used to obtain the representation of a resource, where the resource is specified by a URI and additional request parameters can additionally shape the representation. This has been modelled after the HTTP GET operation and the REST model in general.

In HTTP, a resource is often used to search for information, and existing systems varyingly use the HTTP GET and POST methods to perform a search. Often a POST method is used for the sole reason that a larger set of parameters to the search can be supplied in the request body than can comfortably be transferred in the URI with a GET request. The draft [I-D.snell-search-method] proposes a SEARCH method that is similar to GET in most properties but enables sending a request body as with POST. The FETCH method defined in the present specification is inspired by [I-D.snell-search-method], which updates the definition and semantics of the HTTP SEARCH request method previously defined by [RFC5323]. However, there is no intention to limit FETCH to search-type operations, and the resulting properties may not be the same as those of HTTP SEARCH.

A major problem with GET is that the information that controls the request needs to be bundled up in some unspecified way into the URI. Using the request body for this information has a number of advantages:

- o The client can specify a media type (and a content encoding), enabling the server to unambiguously interpret the request parameters in the context of that media type. Also, the request body is not limited by the character set limitations of URIs, enabling a more natural (and more efficient) representation of certain domain-specific parameters.
- o The request parameters are not limited by the maximum size of the URI. In HTTP, that is a problem as the practical limit for this

size varies. In CoAP, another problem is that the block-wise transfer is not available for transferring large URI options in multiple rounds.

As an alternative to using GET, many implementations make use of the POST method to perform extended requests, even if they are semantically idempotent, safe, and even cacheable, to be able to pass along the input parameters within the request payload as opposed to using the request URI.

The FETCH method provides a solution that spans the gap between the use of GET and POST. As with POST, the input to the FETCH operation is passed along within the payload of the request rather than as part of the request URI. Unlike POST, however the semantics of the FETCH method are more specifically defined.

1.2. PATCH and iPATCH

PATCH is also specified for HTTP in [RFC5789]. Most of the motivation for PATCH described in [RFC5789] also applies here. iPATCH is the idempotent version of PATCH.

The PUT method exists to overwrite a resource with completely new contents, and cannot be used to perform partial changes. When using PUT for partial changes, proxies and caches, and even clients and servers, may get confused as to the result of the operation. PATCH was not adopted in an early design stage of CoAP, however, it has become necessary with the arrival of applications that require partial updates to resources (e.g. [I-D.vanderstok-core-comi]). Using PATCH avoids transferring all data associated with a resource in case of modifications, thereby not burdening the constrained communication medium.

This document relies on knowledge of the PATCH specification for HTTP [RFC5789]. This document provides extracts from [RFC5789] to make independent reading possible.

1.3. Requirements Language

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

1.4. Terminology and Acronyms

This document uses terminology defined in [RFC5789] and [RFC7252].

Specifically, it uses the terms "safe" and "idempotent" as defined in Section 5.1 of [RFC7252]. (Further discussion of safe and idempotent methods can now be found in Section 4.2.1 and 4.2.2 of [RFC7231], respectively; the implications of idempotency of methods on server implementations are also discussed in Section 4.5 of [RFC7252].)

2. FETCH Method

The CoAP FETCH method is used to obtain a representation of a resource, giving a number of request parameters. Unlike the CoAP GET method, which requests that a server return a representation of the resource identified by the effective request URI (as defined by [RFC7252]), the FETCH method is used by a client to ask the server to produce a representation as described by the request parameters (including the request options and the payload) based on the resource specified by the effective request URI. The payload returned in response to a FETCH cannot be assumed to be a complete representation of the resource identified by the effective request URI, i.e., it cannot be used by a cache as a payload to be returned by a GET request.

Together with the request options, the body of the request (which may be constructed from multiple payloads using the block protocol [RFC7959]) defines the request parameters. With the FETCH method, implementations may submit a request body of any media type that is defined with the semantics of selecting information from a resource in such a FETCH request; it is outside the scope of this document how information about media types admissible for the specific resource is obtained by the client (although we can hint that form relations ([I-D.hartke-core-apps]) might be a preferred way). It is RECOMMENDED that any discovery method that allows a client to find out that the server supports FETCH also provides information what FETCH payload media types are applicable.

FETCH requests are both safe and idempotent with regards to the resource identified by the request URI. That is, the performance of a fetch is not intended to alter the state of the targeted resource. (However, while processing a fetch request, a server can be expected to allocate computing and memory resources or even create additional server resources through which the response to the search can be retrieved.)

A successful response to a FETCH request is expected to provide some indication as to the final disposition of the requested operation.

If a successful response includes a body payload, the payload is expected to describe the results of the FETCH operation.

Depending on the response code as defined by [RFC7252], the response to a FETCH request is cacheable; the request body is part of the cache key. Specifically, 2.05 "Content" response codes, the responses for which are cacheable, are a usual way to respond to a FETCH request. (Note that this aspect differs markedly from [I-D.snell-search-method].) (Note also that caches that cannot use the request payload as part of the cache key will not be able to cache responses to FETCH requests at all.) The Max-Age option in the response has equivalent semantics to its use in a GET.

The semantics of the FETCH method change to a "conditional FETCH" if the request message includes an If-Match, or If-None-Match option ([RFC7252]). A conditional FETCH requests that the query be performed only under the circumstances described by the conditional option(s). It is important to note, however, that such conditions are evaluated against the state of the target resource itself as opposed to the results of the FETCH operation.

2.1. Response Codes

FETCH for CoAP adopts the response codes as specified in sections 5.9 and 12.1.2 of [RFC7252] as well as additional response codes mentioned in Section 2.2.

2.2. Error Handling

A FETCH request may fail under certain known conditions. Beyond the conditions already defined in [RFC7252] for GET, noteworthy ones are:

Malformed FETCH payload: If a server determines that the payload provided with a FETCH request is not properly formatted, it can return a 4.00 (Bad Request) CoAP error. The definition of a malformed payload depends upon the CoAP Content-Format specified with the request.

Unsupported FETCH payload: In case a client sends a payload that is inappropriate for the resource identified by the Request-URI, the server can return a 4.15 (Unsupported Content-Format) CoAP error. The server can determine if the payload is supported by checking the CoAP Content-Format specified with the request.

Unprocessable request: This situation occurs when the payload of a FETCH request is determined as valid, i.e. well-formed and supported, however, the server is unable to or incapable of processing the request. The server can return a 4.22

(Unprocessable Entity) CoAP error. In situations when the server has insufficient computing resources to complete the request successfully, it can return a 4.13 (Request Entity Too Large) CoAP error (see also below). In case there are more specific errors that provide more insight into the problem, then those should be used.

Request too large: If the payload of the FETCH request is larger than a CoAP server can process, then it can return the 4.13 (Request Entity Too Large) CoAP error.

It is possible that other error situations, not mentioned here, are encountered by a CoAP server while processing the FETCH request. In these situations other appropriate CoAP status codes can also be returned.

2.3. Option Numbers

FETCH for CoAP adopts the option numbers as specified in sections 5.10 and 12.2 of [RFC7252].

Generally, options defined for GET act in an analogous way for FETCH. Two specific cases are called out in the rest of this section.

2.3.1. The Content-Format Option

A FETCH request MUST include a Content-Format option (see Section 5.10.3 of [RFC7252]) to specify the media type and content encoding of the request body. (Typically, the media type will specifically have been designed to specify details for a selection or a search on a resource.)

2.3.2. The ETag Option

The ETag Option on a FETCH result has the same semantics as defined in Section 5.10.6 of [RFC7252]. In particular, its use as a response option describes the "tagged representation", which for FETCH is the same as the "selected representation". The FETCH payload is input to that selection process and therefore needs to be part of the cache key. Similarly, the use of ETag as a request option can elicit a 2.03 Valid response if the representation associated with the ETag would still be selected by the FETCH request (including its payload).

2.4. Working with Observe

The Observe option [RFC7641] can be used with a FETCH request as it can be used with a GET request.

2.5. Working with Block

The Block1 option [RFC7959] can be used with a FETCH request as it would be used with a POST request; the Block2 option can then be used as with GET or POST.

2.6. Building FETCH Requests

One property of FETCH that may be non-obvious is that a FETCH request cannot be generated from a link alone, but also needs a way to generate the request payload. Again, form relations ([I-D.hartke-core-apps]) may be able to fill parts of this gap.

2.7. A Simple Example for FETCH

The FETCH method needs a media type for its payload (as expressed by the Content-Format request option) that specifies the search query in a similar detail as is shown for the patch payload in the PATCH example in Section 3.1. ([I-D.snell-search-method] invents a "text/query" format based on some hypothetical SQL dialect for its examples.)

The example below illustrates retrieval of a subset of a JSON [RFC7159] object (the same object as used in Section 3.1). Using a hypothetical media type "application/example-map-keys+json" (with a Content-Format ID of NNN – not defined as this is just an example), the client specifies the items in the object that it wants: it supplies a JSON array giving the map keys for these items. A resource located at "coap://www.example.com/object" can be represented by a JSON document that we will consider as the target of the FETCH. The client wants to learn the contents of the single map key "foo" within this target:

```
{
  "x-coord": 256,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

FETCH example: JSON document returned by GET

The example FETCH request specifies a single top-level member desired by giving its map key as the sole element of the "example-map-keys" payload:


```
FETCH CoAP://www.example.com/object
Content-Format: NNN (application/example-map-keys+json)
Accept: application/json
[
  "foo"
]
```

FETCH example: Request

The server returns a subset document with just the selected member:

```
2.05 Content
Content-Format: 50 (application/json)
{
  "foo": ["bar","baz"]
}
```

FETCH example: Response with subset JSON document

By the logic of this example, the requester could have entered more than one map key into the request payload array and would have received a more complete subset of the top-level JSON object that is representing the resource.

3. PATCH and iPATCH Methods

The PATCH and iPATCH methods request that a set of changes described in the request payload is applied to the target resource of the request. The set of changes is represented in a format identified by a media type. If the Request-URI does not point to an existing resource, the server MAY create a new resource with that URI, depending on the patch document type (whether it can logically modify a null resource) and permissions, as well as other conditions such as the degree of control the server gives clients in creating new entries in its URI space (see also Section 3.4). Creation of a new resource would result in a 2.01 (Created) Response Code dependent on the patch document type.

Restrictions to a PATCH or iPATCH request can be made by including the If-Match or If-None-Match options in the request (see Section 5.10.8.1 and 5.10.8.2 of [RFC7252]). If the resource could not be created or modified, then an appropriate Error Response Code SHOULD be sent.

The difference between the PUT and PATCH requests is documented in [RFC5789]. When a request is intended to effect a partial update of a given resource, clients cannot use PUT while supplying just the update, but might be able to use PATCH or iPATCH.

The PATCH method is not safe and not idempotent, as with the HTTP PATCH method specified in [RFC5789].

The iPATCH method is not safe but idempotent, as with the CoAP PUT method specified in [RFC7252], Section 5.8.3.

A client can mark a request as idempotent by using the iPATCH method instead of the PATCH method. This is the only difference between the two. The indication of idempotence may enable the server to keep less state about the interaction; some constrained servers may only implement the iPATCH variant for this reason.

PATCH and iPATCH are both atomic. The server MUST apply the entire set of changes atomically and never provide a partially modified representation to a concurrently executed GET request. Given the constrained nature of the servers, most servers will only execute CoAP requests consecutively, thus preventing a concurrent partial overlapping of request modifications. Resuming, modifications MUST NOT be applied to the server state when an error occurs or only a partial execution is possible on the resources present in the server.

The atomicity applies to a single server. When a PATCH or iPATCH request is multicast to a set of servers, each server can either execute all required modifications or not. It is not required that all servers execute all modifications or none. An Atomic Commit protocol that provides multiple server atomicity is out of scope.

A PATCH or iPATCH response can invalidate a cache as with the PUT response. Caching behaviour as function of the successful (2.xx) response codes for PATCH or iPATCH are:

- o A 2.01 (Created) response invalidates any cache entry for the resource indicated by the Location-* Options; the payload is a representation of the action result.
- o A 2.04 (Changed) response invalidates any cache entry for the target resource; the payload is a representation of the action result.

There is no guarantee that a resource can be modified with PATCH or iPATCH. Servers MUST ensure that a received PATCH body is appropriate for the type of resource identified by the target resource of the request.

It is RECOMMENDED that any discovery method that allows a client to find out that the server supports one of PATCH and iPATCH also provides information what patch payload media types are applicable

and which of the two methods are implemented by the server for each of these media types.

Servers that do not rely on the idempotency of iPATCH can easily support both PATCH and iPATCH, and it is RECOMMENDED they do so. This is inexpensive to do, as, for iPATCH, there is no requirement on the server to check that the client's intention that the request be idempotent is fulfilled (although there is diagnostic value in that check, so a less-constrained implementation may want to perform it).

3.1. Simple Examples for PATCH and iPATCH

The example is taken over from [RFC6902], which specifies a JSON notation for PATCH operations. A resource located at `coap://www.example.com/object` contains a target JSON document.

JSON document original state:

```
{
  "x-coord": 256,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

REQ: iPATCH CoAP://www.example.com/object
Content-Format: 51 (application/json-patch+json)

```
[
  { "op": "replace", "path": "x-coord", "value": 45 }
]
```

RET: CoAP 2.04 Changed

JSON document final state:

```
{
  "x-coord": 45,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

This example illustrates use of an idempotent modification to the `x-coord` member of the existing resource `"object"`. The 2.04 (Changed) response code is conform with the CoAP PUT method.

The same example using the Content-Format `application/merge-patch+json` from [RFC7396] looks like:

JSON document original state:

```
{
  "x-coord": 256,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

```
REQ: iPATCH CoAP://www.example.com/object
Content-Format: 52 (application/merge-patch+json)
{ "x-coord":45}
```

RET: CoAP 2.04 Changed

JSON document final state:

```
{
  "x-coord": 45,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

The examples show the use of the iPATCH method, but the use of the PATCH method would have led to the same result. Below a non-idempotent modification is shown. Because the action is non-idempotent, iPATCH returns an error, while PATCH executes the action.

JSON document original state:

```
{
  "x-coord": 256,
  "y-coord": 45,
  "foo": ["bar", "baz"]
}
```

REQ: iPATCH CoAP://www.example.com/object
Content-Format: 51 (application/json-patch+json)

```
[
  { "op": "add", "path": "foo/1", "value": "bar" }
]
```

RET: CoAP 4.00 Bad Request
Diagnostic payload: Patch format not idempotent

JSON document final state is unchanged

REQ: PATCH CoAP://www.example.com/object
Content-Format: 51 (application/json-patch+json)

```
[
  { "op": "add", "path": "foo/1", "value": "bar" }
]
```

RET: CoAP 2.04 Changed

JSON document final state:

```
{
  "x-coord": 45,
  "y-coord": 45,
  "foo": ["bar", "bar", "baz"]
}
```

3.2. Response Codes

PATCH and iPATCH for CoAP adopt the response codes as specified in sections 5.9 and 12.1.2 of [RFC7252] and add 4.09 "Conflict" and 4.22 "Unprocessable Entity" with the semantics specified in Section 3.4 of the present specification.

3.3. Option Numbers

PATCH and iPATCH for CoAP adopt the option numbers as specified in sections 5.10 and 12.2 of [RFC7252].

3.4. Error Handling

A PATCH or iPATCH request may fail under certain known conditions. These situations should be dealt with as expressed below.

Malformed PATCH or iPATCH payload: If a server determines that the payload provided with a PATCH or iPATCH request is not properly formatted, it can return a 4.00 (Bad Request) CoAP error. The definition of a malformed payload depends upon the CoAP Content-Format specified with the request.

Unsupported PATCH or iPATCH payload: In case a client sends a payload that is inappropriate for the resource identified by the Request-URI, the server can return a 4.15 (Unsupported Content-Format) CoAP error. The server can determine if the payload is supported by checking the CoAP Content-Format specified with the request.

Unprocessable request: This situation occurs when the payload of a PATCH request is determined as valid, i.e. well-formed and supported, however, the server is unable to or incapable of processing the request. The server can return a 4.22 (Unprocessable Entity) CoAP error. More specific scenarios might include situations when:

- * the server has insufficient computing resources to complete the request successfully -- 4.13 (Request Entity Too Large) CoAP Response Code (see below),
- * the resource specified in the request becomes invalid by applying the payload -- 4.09 (Conflict) CoAP Response Code (see below)).

In case there are more specific errors that provide more insight into the problem, then those should be used.

Resource not found: The 4.04 (Not Found) error should be returned in case the payload of a PATCH request cannot be applied to a non-existent resource.

Failed precondition: In case the client uses the conditional If-Match or If-None-Match option to define a precondition for the PATCH request, and that precondition fails, then the server can return the 4.12 (Precondition Failed) CoAP error.

Request too large: If the payload of the PATCH request is larger than a CoAP server can process, then it can return the 4.13 (Request Entity Too Large) CoAP error.

Conflicting state: If the modification specified by a PATCH or iPATCH request causes the resource to enter an inconsistent state that the server cannot resolve, the server can return the 4.09 (Conflict) CoAP response. The server SHOULD generate a payload

that includes enough information for a user to recognize the source of the conflict. The server MAY return the actual resource state to provide the client with the means to create a new consistent resource state. Such a situation might be encountered when a structural modification is applied to a configuration data-store, but the structures being modified do not exist.

Concurrent modification: Resource constrained devices might need to process requests in the order they are received. In case requests are received concurrently to modify the same resource but they cannot be queued, the server can return a 5.03 (Service unavailable) CoAP response code.

Conflict handling failure: If the modification implies the reservation of resources or the waiting on conditions to become true, leading to a too long request execution time, the server can return 5.03 (service unavailable) response code.

It is possible that other error situations, not mentioned here, are encountered by a CoAP server while processing the PATCH request. In these situations other appropriate CoAP status codes can also be returned.

4. The New Set of CoAP Methods

Adding three new methods to CoAP's existing four may seem like a major change. However, both FETCH and the two PATCH variants fit well into the REST paradigm and have been anticipated on the HTTP side. Adding both a non-idempotent and an idempotent PATCH variant allows to keep interoperability with HTTP's PATCH method as well as the use/indication of an idempotent PATCH if that is possible, saving significant effort on the server side.

Interestingly, the three new methods fit into the old table of methods with a surprising similarity in the idempotence and safety attributes:

Code	Name	Code	Name	safe	idempotent
0.01	GET	0.05	FETCH	yes	yes
0.02	POST	0.06	PATCH	no	no
0.03	PUT	0.07	iPATCH	no	yes
0.04	DELETE			no	yes

5. Security Considerations

This section analyses the possible threats to the CoAP FETCH and PATCH or iPATCH methods. It is meant to inform protocol and application developers about the security limitations of CoAP FETCH and PATCH or iPATCH as described in this document.

The FETCH method is subject to the same general security considerations as all CoAP methods as described in Section 11 of [RFC7252]. Specifically, the security considerations for FETCH are closest to those of GET, except that the FETCH request carries a payload that may need additional protection. The payload of a FETCH request may reveal more detailed information about the specific portions of a resource of interest to the requester than a GET request for the entire resource would; this may mean that confidentiality protection of the request by DTLS or other means is needed for FETCH where it wouldn't be needed for GET.

The PATCH and iPATCH methods are subject to the same general security considerations as all CoAP methods as described in Section 11 of [RFC7252]. The specific security considerations for PATCH or iPATCH are nearly identical to the security considerations for PUT ([RFC7252]); the security considerations of Section 5 of [RFC5789] also apply to PATCH and iPATCH. Specifically, there is likely to be a need for authorizing requests (possibly through access control and/or authentication) and for ensuring that data is not corrupted through transport errors or through accidental overwrites. The mechanisms used for PUT can be used for PATCH or iPATCH as well.

The new methods defined in the present specification are secured following the CoAP recommendations for the existing methods as specified in section 9 of [RFC7252]. When additional security techniques are standardized for CoAP (e.g., Object Security), these are then also available for securing the new methods.

6. IANA Considerations

IANA is requested to add the following entries to the sub-registry "CoAP Method Codes":

Code	Name	Reference
0.05	FETCH	[RFCthis]
0.06	PATCH	[RFCthis]
0.07	iPATCH	[RFCthis]

The FETCH method is idempotent and safe, and it returns the same response codes that GET can return, plus 4.13 (Request Entity Too Large), 4.15 (Unsupported Content-Format), and 4.22 (Unprocessable Entity) with the semantics specified in Section 2.2.

The PATCH method is neither idempotent nor safe. It returns the same response codes that POST can return, plus 4.09 (Conflict) and 4.22 (Unprocessable Entity) with the semantics specified in Section 3.4.

The iPATCH method is identical to the PATCH method, except that it is idempotent.

IANA is requested to add the following code to the sub-registry "CoAP response codes":

Code	Name	Reference
4.09	Conflict	[RFCthis]
4.22	Unprocessable Entity	[RFCthis]

IANA is requested to add entries to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry:

Media Type	Encoding	ID	Reference
application/json-patch+json		51	[RFC6902]
application/merge-patch+json		52	[RFC7396]

Editors' note (to be removed by RFC editor): RFC 6902 and RFC 7396 are not necessary to implement the present specification, not even an option for it. It was just convenient to use the present document to register content-format identifiers for them (and they are used in examples). Therefore, these references are correctly classified as informative.

7. Change log

When published as an RFC, this section needs to be removed.

Version 00 is a composition from draft-vanderstok-core-patch-03 and draft-bormann-core-coap-fetch-00 and replaces these two drafts.

Version 01 added an example for FETCH and is more explicit about some response codes and options.

Version 02 addresses the WGLC comments.

Version 03 addresses the IETF last-call comments.

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, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, DOI 10.17487/RFC5789, March 2010, <<http://www.rfc-editor.org/info/rfc5789>>.
- [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, <<http://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, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc7959>>.

8.2. Informative References

- [RFC5323] Reschke, J., Ed., Reddy, S., Davis, J., and A. Babich, "Web Distributed Authoring and Versioning (WebDAV) SEARCH", RFC 5323, DOI 10.17487/RFC5323, November 2008, <<http://www.rfc-editor.org/info/rfc5323>>.
- [RFC6902] Bryan, P., Ed. and M. Nottingham, Ed., "JavaScript Object Notation (JSON) Patch", RFC 6902, DOI 10.17487/RFC6902, April 2013, <<http://www.rfc-editor.org/info/rfc6902>>.

- [RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <<http://www.rfc-editor.org/info/rfc7159>>.
- [RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014, <<http://www.rfc-editor.org/info/rfc7396>>.
- [I-D.vanderstok-core-comi]
Stok, P., Bierman, A., Veillette, M., and A. Pelov, "CoAP Management Interface", draft-vanderstok-core-comi-10 (work in progress), October 2016.
- [I-D.hartke-core-apps]
Hartke, K., "CoRE Application Descriptions", draft-hartke-core-apps-05 (work in progress), October 2016.
- [I-D.snell-search-method]
Reschke, J., Malhotra, A., and J. Snell, "HTTP SEARCH Method", draft-snell-search-method-00 (work in progress), April 2015.

Acknowledgements

Klaus Hartke has pointed out some essential differences between CoAP and HTTP concerning PATCH, and found a number of problems in an earlier version of Section 2. We are grateful for discussions with Christian Amsuss, Andy Bierman, Timothy Carey, Paul Duffy, Matthias Kovatsch, Michel Veillette, Michael Verschoor, Thomas Watteyne, and Gengyu Wei. Christian Groves provided detailed comments during the Working-Group Last Call, and Christer Holmberg's Gen-ART review provided some further editorial improvement. Further last-call reviews were provided by Sheng Jiang and Phillip Hallam-Baker. As usual, the IESG had some very good reviews, we would like to specifically call out those by Alexey Melnikov (responsible AD) and Alissa Cooper.

Authors' Addresses

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

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

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

Anuj Sehgal
Consultant

Email: anuj@iurs.org

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

Z. Shelby
ARM
M. Koster
SmartThings
C. Groves

J. Zhu
Huawei
B. Silverajan, Ed.
Tampere University
March 11, 2019

Reusable Interface Definitions for Constrained RESTful Environments
draft-ietf-core-interfaces-14

Abstract

This document defines a set of Constrained RESTful Environments (CoRE) Link Format Interface Descriptions [RFC6690] applicable for use in constrained environments. These include the: Actuator, Parameter, Read-only parameter, Sensor, Batch, Linked Batch and Link List interfaces.

The Batch, Linked Batch and Link List interfaces make use of resource collections. This document further describes how collections relate to interfaces.

Many applications require a set of interface descriptions in order provide the required functionality. This document defines an Interface Description attribute value to describe resources conforming to a particular interface.

Editor's notes:

- o The git repository for the draft is found at <https://github.com/core-wg/interfaces>

Status of This Memo

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

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

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

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

Copyright Notice

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

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

Table of Contents

1. Introduction	3
2. Terminology	4
3. Collections	4
3.1. Introduction to Collections	5
3.2. Use Cases for Collections	5
3.3. Collection Types	6
3.4. Content-Formats for Collections	6
3.5. Link Embedding	7
3.6. Links and Items in Collections	7
3.7. Queries on Collections	8
3.8. Observing Collections	8
4. Interface Descriptions	9
4.1. Link List	11
4.2. Batch	11
4.3. Linked Batch	12
4.4. Sensor	13
4.5. Parameter	14
4.6. Read-only Parameter	14
4.7. Actuator	14
5. Security Considerations	15
6. IANA Considerations	15
6.1. Link List	15
6.2. Batch	16
6.3. Linked Batch	16

6.4. Sensor	16
6.5. Parameter	17
6.6. Read-only parameter	17
6.7. Actuator	17
7. Acknowledgements	17
8. Contributors	18
9. Changelog	18
10. References	22
10.1. Normative References	22
10.2. Informative References	22
Appendix A. Current Usage of Interfaces	23
A.1. Constrained RESTful Environments (CoRE) Link Format (IETF)	23
A.2. Open Connectivity Foundation (OCF)	24
Authors' Addresses	24

1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces. CoRE Link-format is a standard for doing Web Linking [RFC8288] in constrained environments. SenML [RFC8428] is a simple data model and representation format for composite and complex structured resources. CoRE Link-Format and SenML can be used by CoAP [RFC7252] or HTTP servers.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop. Machine application clients must be able to adapt to different resource organizations without advance knowledge of the specific data structures hosted by each connected thing. The use of Web Linking for the description and discovery of resources hosted by constrained origin servers is specified by CoRE Link Format [RFC6690]. CoRE Link Format additionally defines a link attribute for interface description ("if") that can be used to describe the REST interface of a resource, and may include a link to a description document.

This document defines a set of Link Format interface descriptions for some common design patterns that enable the server side composition and organization, and client side discovery and consumption, of machine resources using Web Linking. A client discovering the "if" link attribute will be able to consume resources based on its knowledge of the expected interface types. In this sense the Interface Type acts in a similar way as a Content-Format, but as a selector for a high level functional abstraction.

An interface description describes a resource in terms of its associated content formats, data types, URI templates, REST methods, parameters, and responses. Basic interface descriptions are defined for sensors, and actuators.

A set of collection types is defined for organizing resources for discovery, and for various forms of bulk interaction with resource sets using typed embedding links.

This document first defines the concept of collection interface descriptions. It then defines a number of generic interface descriptions that may be used in constrained environments. Several of these interface descriptions utilise collections.

Whilst this document assumes the use of CoAP [RFC7252], the REST interfaces described can also be realized using HTTP [RFC7230].

2. Terminology

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

This document requires readers to be familiar with all the terms and concepts that are discussed in [RFC8288] and [RFC6690]. This document makes use of the following additional terminology:

Gradual Reveal: A REST design where resources are discovered progressively using Web Linking.

Interface Description: The Interface Description describes the generic REST interface to interact with a resource or a set of resources. Its use is described via the Interface Description 'if' attribute which is an opaque string used to provide a name or URI indicating a specific interface definition used to interact with the target resource. One can think of this as describing verbs usable on a resource.

Resource Discovery: The process allowing a client to identify resources being hosted on an origin server.

3. Collections

3.1. Introduction to Collections

A Collection is a resource which represents one or more related resources. [RFC6573] describes the "item" and "collection" Link Relation. An "item" link relation identifies a member of collection. A "collection" indicates the collection that an item is a member of. For example, a collection might be a resource representing a catalog of products, while an item is a resource related to an individual product.

Section 1.2.2/[RFC6690] also describes resource collections.

This document uses the concept of "collection" and applies it to interface descriptions. A collection interface description consists of a set of links and a set of items pointed to by the links which may be sub-resources of the collection resource. The collection interface descriptions described in this document are Link List, Batch and Linked Batch.

The links in a collection are represented in CoRE Link-Format Content-Formats including JSON and CBOR variants, and the items in the collection may be represented by SenML, including JSON and CBOR variants. In general, a collection may support items of any available Content-Format.

A particular resource item may be a member of more than one collection at a time by being linked to, but may only be a subresource of one collection.

Some collections may have pre-configured items and links, and some collections may support dynamic creation and removal of items and links. Likewise, modification of items in some collections may be permitted, and not in others.

Links in collections may be selected for processing by a particular request by using Query Filtering as described in CoRE Link-Format [RFC6690].

3.2. Use Cases for Collections

Collections may be used to provide gradual reveal of resources on an endpoint. There may be a small set of links at the .well-known/core location, which may in turn point to other collections of resources that represent device information, device configuration, device management, and various functional clusters of resources on the device.

A collection may be used to group a set of like resources for bulk state update or actuation. For example, the brightness control resources of a number of luminaries may be grouped by linking to them in a collection. The collection type may support receiving a single update from a client and sending that update to each resource item in the collection.

Items may be sub-resources of the collection resource. This enables updates to multiple items in the collection to be processed together within the context of the collection resource.

3.3. Collection Types

There are three collection types defined in this document:

Collection Type	if=
Link List	core.ll
Batch	core.b
Linked Batch	core.lb

Table 1: Collection Type Summary

The interface description defined in this document offer a deeper explanation of the methods that may be applied to the three collections.

3.4. Content-Formats for Collections

The collection interfaces can use the CoRE Link-Format for the link representations and SenML or text/plain for representations of items. The examples given are for collections that expose resources and links in these formats.

The choice of whether to return a representation of the links or of the items or of the collection format is determined by the Accept header option in the request. Likewise, the choice of updating link metadata or item data or the collection resource itself is determined by the Content-Format option in the header of the update request operation.

The default Content-Formats for collection types described in this document are:

Links: application/link-format, application/link-format+json

Items: application/senml+json, text/plain

3.5. Link Embedding

Collections may provide resource encapsulation by supporting link embedding. Link embedding may be used to provide a single resource with which a client may interact to obtain a set of related resource values. This is analogous to an image tag (link) causing the image to display inline in a browser window. Link embedding enables the bulk processing of items in the collection using a single operation targeting the collection resource. Performing a GET on a collection resource may return a single representation containing all of the embedded linked resources. For example, a collection for manufacturer parameters may consist of manufacturer name, date of manufacture, location of manufacture, and serial number resources which can be read as a single SenML data object.

A subset of resources in the collection may be selected for operation using Query Filtering. Bulk Read operations using GET return a SenML representation of all selected resources. Bulk item Update operations using PUT or POST apply the payload document to all selected resource items in the collection. A Batch update is performed by applying the resource values in the payload document to all resources in the collection that match any resource name in the payload document.

3.6. Links and Items in Collections

Links use CoRE Link-Format representation by default and may point to any resource reachable from the context of the collection. This includes links to resources with absolute paths as well as links that point to other network locations, if the context of the collection allows. Links to sub-resources in the collection MUST have a path-element starting with the resource name, as per [RFC3986]. Links to resources in the global context MUST start with a root path identifier [RFC8288]. Links to other collections are formed per [RFC3986].

Examples of links:

</sen/>;if="core.lb": Link to the /sen/ collection describing it as a core.lb type collection (Linked Batch)

</sen/temp>;rt="temperature": A link to the temp resource with an absolute path.

`<temp>;rt="temperature"`: Link to the temp subresource of the collection in which this link appears.

`<temp>;anchor="/sen/"`: A link to the temp subresource of the collection /sen/ which is assumed not to be a subresource of the collection in which the link appears, but is expected to be identified in the collection by resource name.

Links in the collection MAY be Read, Updated, Added, or Removed using the CoRE Link-Format or JSON Merge-Patch Content-Formats on the collection resource. Reading links uses the GET method and returns an array or list containing the link-values of all selected links. Links may be added to the collection using POST or PATCH methods. Updates to links MUST use the PATCH method and MAY use query filtering to select links for updating. The PATCH method on links MUST use the JSON Merge-Patch Content-Format (application/merge-patch+json) specified in [RFC7396].

Items in the collection SHOULD be represented using the SenML (application/senml+json) or plain text (text/plain) Content-Formats, depending on whether the representation is of a single data point or multiple data points. Items MAY be represented using any supported Content-Format.

3.7. Queries on Collections

Collections MAY support query filtering as defined in CoRE Link-Format [RFC6690]. Operations targeting either the links or the items MAY select a subset of links and items in the collection by using query filtering. The Content-Format specified in the request header selects whether links or items are targeted by the operation.

3.8. Observing Collections

Resource Observation via [I-D.ietf-core-dynlink] using CoAP [RFC7252] MAY be supported on items in a collection. A subset of the conditional observe parameters MAY be specified to apply. In most cases pmin and pmax are useful. Resource observation on a collection's resource returns the collection representation. Observation Responses, or notifications, SHOULD provide the collection representations in SenML Content-Format. Notifications MAY include multiple observations of the collection resource, with SenML time stamps indicating the observation times.

4. Interface Descriptions

This section defines REST interfaces for Sensor, Parameter, Read-Only Parameter and Actuator resource types, in addition to the Link List, Batch and Linked Batch collection types. Each type is described along with its Interface Description attribute value, valid methods and content formats. These are shown for each interface in the table below.

The if= column defines the Interface Description (if=) attribute value to be used in the CoRE Link Format for a resource conforming to that interface. When this value appears in the if= attribute of a link, the resource MUST support the corresponding REST interface described in this section. The resource MAY support additional functionality, which is out of scope for this document. Although these interface descriptions are intended to be used with the CoRE Link Format, they are applicable for use in any REST interface definition.

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

Interface	if=	Methods	Content-Formats
Link List	core.ll	GET	link-format
Batch	core.b	GET, PUT, POST	senml
Linked Batch	core.lb	GET, PUT, POST, DELETE	link-format, senml
Sensor	core.s	GET	senml, text/plain
Parameter	core.p	GET, PUT	senml, text/plain
Read-only Parameter	core.rp	GET	senml, text/plain
Actuator	core.a	GET, PUT, POST	senml, text/plain

Table 2: Interface Description Summary

The following is an example of links in the CoRE Link Format using these interface descriptions.

```

Req: GET /.well-known/core
Res: 2.05 Content (application/link-format)
</s/>;rt="simple.sen";if="core.b",
</s/light>;rt="simple.sen.lt";if="core.s",
</s/temp>;rt="simple.sen.tmp";if="core.s";obs,
</s/humidity>;rt="simple.sen.hum";if="core.s",
</a/>;rt="simple.act";if="core.b",
</a/1/led>;rt="simple.act.led";if="core.a",
</a/2/led>;rt="simple.act.led";if="core.a",
</d/>;rt="simple.dev";if="core.ll",
</l/>;if="core.lb"

```

Figure 1: Binding Interface Example

4.1. Link List

Link List is the base interface to provide gradual reveal of resources on a CoRE origin server. It is used to retrieve (GET) a list of resources on an origin server. The GET request SHOULD contain an Accept option with the application/link-format content format. However if the resource does not support any other form of content-format the Accept option MAY be elided.

Note: The use of an Accept option with application/link-format is recommended even though it is not strictly needed for the Link List interface because this interface is extended by the batch and linked batch interfaces where different content-formats are possible.

The request returns a list of URI references with absolute paths to the resources as defined in CoRE Link Format. This interface is typically used with a parent resource to enumerate sub-resources but may be used to reference any resource on an origin server.

The following example interacts with a Link List /d/ containing Parameter sub-resources /d/name, /d/model.

```
Req: GET /d/ (Accept:application/link-format)
Res: 2.05 Content (application/link-format)
</d/name>;rt="simple.dev.n";if="core.p",
</d/model>;rt="simple.dev.mdl";if="core.rp"
```

4.2. Batch

The Batch interface is used to manipulate a collection of sub-resources at the same time. The Batch interface description supports the same methods as its sub-resources, and can be used to read (GET), update (PUT) or apply (POST) the values of those sub-resource with a single resource representation. The sub-resources of a Batch MAY be heterogeneous. Hence, a method used on the Batch only applies to sub-resources that support it. For example Sensor interfaces do not support PUT, and thus a PUT request to a Sensor member of that Batch would be ignored. A batch requires the use of SenML Media types in order to support multiple sub-resources.

The following example interacts with a Batch /s/ with Sensor sub-resources /s/light, /s/temp and /s/humidity.

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

4.3. Linked Batch

The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the origin server, a Linked Batch is dynamically controlled by a client. A Linked Batch resource has no sub-resources. Instead the resources forming the batch are referenced using Web Linking [RFC8288] and the CoRE Link Format [RFC6690]. A request with a POST method and a content format of application/link-format simply appends new resource links to the collection. The links in the payload MUST reference a resource on the origin server with an absolute path. A DELETE request removes the entire collection. All other requests available for a basic Batch are still valid for a Linked Batch.

The following example interacts with a Linked Batch /l/ and creates a collection containing /s/light, /s/temp and /s/humidity in 2 steps.


```
Req: POST /1/ (Content-Format: application/link-format)
</s/light>,</s/temp>
Res: 2.04 Changed
```

```
Req: GET /1/
Res: 2.05 Content (application/senml+json)
[
  { "bn": "example.com/" },
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "Cel" }
]
```

```
Req: POST /1/ (Content-Format: application/link-format)
</s/humidity>
Res: 2.04 Changed
```

```
Req: GET /1/ (Accept: application/link-format)
Res: 2.05 Content (application/link-format)
</s/light>,</s/temp>,</s/humidity>
```

```
Req: GET /1/
Res: 2.05 Content (application/senml+json)
[
  { "bn": "example.com/" },
  { "n": "/s/light", "v": 123, "u": "lx" },
  { "n": "/s/temp", "v": 27.2, "u": "Cel" },
  { "n": "/s/humidity", "v": 80, "u": "%RH" }
]
```

```
Req: DELETE /1/
Res: 2.02 Deleted
```

4.4. Sensor

The Sensor interface allows the value of a sensor resource to be read (GET). The Media type of the resource can be either plain text or SenML. Plain text MAY be used for a single measurement that does not require meta-data. For a measurement with meta-data such as a unit or time stamp, SenML SHOULD be used. A resource with this interface MAY use SenML to return multiple measurements in the same representation, for example a list of recent measurements.

The following are examples of Sensor interface requests in both text/plain and application/senml+json.

```
Req: GET /s/humidity (Accept: text/plain)
Res: 2.05 Content (text/plain)
80
```

```
Req: GET /s/humidity (Accept: application/senml+json)
Res: 2.05 Content (application/senml+json)
[
  { "bn": "example.com/s/" },
  { "n": "humidity", "v": 80, "u": "%RH" }
]
```

4.5. Parameter

The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read (GET) or update (PUT). Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading and updating a parameter.

```
Req: GET /d/name
Res: 2.05 Content (text/plain)
node5
```

```
Req: PUT /d/name (text/plain)
outdoor
Res: 2.04 Changed
```

4.6. Read-only Parameter

The Read-only Parameter interface allows configuration parameters to be read (GET) but not updated. Plain text or SenML Media types MAY be returned from this type of interface.

The following example shows request for reading such a parameter.

```
Req: GET /d/model
Res: 2.05 Content (text/plain)
SuperNode200
```

4.7. Actuator

The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include for example LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read (GET) or the actuator value can be updated

(PUT). In addition, this interface allows the use of POST to change the state of an actuator, for example to toggle between its possible values. Plain text or SenML Media types MAY be returned from this type of interface. A resource with this interface MAY use SenML to include multiple measurements in the same representation, for example a list of recent actuator values or a list of values to updated.

The following example shows requests for reading, setting and toggling an actuator (turning on a LED).

```
Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0
```

```
Req: PUT /a/1/led (text/plain)
1
Res: 2.04 Changed
```

```
Req: POST /a/1/led (text/plain)
Res: 2.04 Changed
```

```
Req: GET /a/1/led
Res: 2.05 Content (text/plain)
0
```

5. Security Considerations

An implementation of a client needs to be prepared to deal with responses to a request that differ from what is specified in this document. A server implementing what the client thinks is a resource with one of these interface descriptions could return malformed representations and response codes either by accident or maliciously. A server sending maliciously malformed responses could attempt to take advantage of a poorly implemented client for example to crash the node or perform denial of service. Conversely, a malicious client could attempt to write to arbitrary resources on a poorly implemented server described in a linked batch.

6. IANA Considerations

This document registers the following CoRE Interface Description (if=) Link Target Attribute Values.

6.1. Link List

Attribute Value: core.ll

Description: The Link List interface is used to retrieve a list of resources on an origin server.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.2. Batch

Attribute Value: core.b

Description: The Batch interface is used to manipulate a collection of sub-resources at the same time.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.3. Linked Batch

Attribute Value: core.lb

Description: The Linked Batch interface is an extension of the Batch interface. Contrary to the basic Batch which is a collection statically defined by the origin server, a Linked Batch is dynamically controlled by a client.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.4. Sensor

Attribute Value: core.s

Description: The Sensor interface allows the value of a sensor resource to be read.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.5. Parameter

Attribute Value: core.p

Description: The Parameter interface allows configurable parameters and other information to be modeled as a resource. The value of the parameter can be read or update.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.6. Read-only parameter

Attribute Value: core.rp

Description: The Read-only Parameter interface allows configuration parameters to be read but not updated.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

6.7. Actuator

Attribute Value: core.a

Description: The Actuator interface is used by resources that model different kinds of actuators (changing its value has an effect on its environment). Examples of actuators include LEDs, relays, motor controllers and light dimmers. The current value of the actuator can be read or the actuator value can be updated. In addition, this interface allows the use of POST to change the state of an actuator, for example to toggle between its possible values.

Reference: This document. Note to RFC Editor - please insert the appropriate RFC reference.

Notes: None

7. Acknowledgements

Acknowledgement is given to colleagues from the SENSEI project who were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further

requirements for interface descriptions have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this document. Ari Keraenen provided updated SenML examples. Christian Amsuss supplied a comprehensive review of draft -12.

8. Contributors

Matthieu Vial
Schneider-Electric
Grenoble
France

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

9. Changelog

Changes from -13 to -14:

- o Version update, with changes in editor's contact information

Changes from -12 to -13:

- o SenML examples now use the Base Name (bn) labels from RFC 8428
- o Security considerations discusses client misuse of linked batches

Changes from -11 to -12:

- o Removed all text referring to function sets/profiles
- o Clarified list collections
- o Content-formats for collections and items rectified
- o Simplified Appendix A and removed Appendix B

Changes from -10 to -11:

- o Added a new Section 3.4 for Link Embedding
- o Updated examples in Section 3.5
- o Removed "Service Discovery" from Terminologies
- o Removed discussion of function sets

Changes from -09 to -10:

- o Section 1: Amendments to remove discussing properties. *
- o New author and editor added.

Changes from -08 to -09:

- o Section 3.6: Modified to indicate that the entire collection resource is returned.
- o General: Added editor's note with open issues.

Changes from -07 to -08:

- o Section 3.3: Modified Accepts to Accept header option.
- o Addressed the editor's note in Section 4.1 to clarify the use of the Accept option.

Changes from -06 to -07:

- o Corrected Figure 1 sub-resource names e.g. tmp to temp and hum to humidity.
- o Addressed the editor's note in Section 4.2.
- o Removed section on function sets and profiles as agreed to at the IETF#97.

Changes from -05 to -06:

- o Updated the abstract.
- o Section 1: Updated introduction.
- o Section 2: Alphabetised the order
- o Section 2: Removed the collections definition in favour of the complete definition in the collections section.
- o Removed section 3 on interfaces in favour of an updated definition in section 1.3.
- o General: Changed interface type to interface description as that is the term defined in RFC6690.
- o Removed section on future interfaces.

- o Section 8: Updated IANA considerations.
- o Added Appendix A to discuss current state of the art wrt to collections, function sets etc.

Changes from -04 to -05:

- o Removed Link Bindings and Observe attributes. This functionality is now contained in I-D.ietf-core-dynlink.
- o Hypermedia collections have been removed. This is covered in a new T2TRG draft.
- o The WADL description has been removed.
- o Fixed minor typos.
- o Updated references.

Changes from -03 to -04:

- o Fixed tickets #385 and #386.
- o Changed abstract and intro to better describe content.
- o Focus on Interface and not function set/profiles in intro.
- o Changed references from draft-core-observe to RFC7641.
- o Moved Function sets and Profiles to section after Interfaces.
- o Moved Observe Attributes to the Link Binding section.
- o Add a Collection section to describe the collection types.
- o Add the Hypermedia Collection Interface Description.

Changes from -02 to -03:

- o Added lt and gt to binding format section.
- o Added pmin and pmax observe parameters to Observation Attributes.
- o Changed the definition of lt and gt to limit crossing.
- o Added definitions for getattr and setattr to WADL.
- o Added getattr and setattr to observable interfaces.

- o Removed query parameters from Observe definition.
- o Added observe-cancel definition to WADL and to observable interfaces.

Changes from -01 to -02:

- o Updated the date and version, fixed references.
- o "Removed pmin and pmax observe parameters "[Ticket #336]"."

Changes from -00 to WG Document -01

- o Improvements to the Function Set section.

Changes from -05 to WG Document -00

- o Updated the date and version.

Changes from -04 to -05

- o Made the Observation control parameters to be treated as resources rather than Observe query parameters. Added Less Than and Greater Than parameters.

Changes from -03 to -04

- o Draft refresh

Changes from -02 to -03

- o Added Bindings
- o Updated all rt= and if= for the new Link Format IANA rules

Changes from -01 to -02

- o Defined a Function Set and its guidelines.
- o Added the Link List interface.
- o Added the Linked Batch interface.
- o Improved the WADL interface definition.
- o Added a simple profile example.

10. References

10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

10.2. Informative References

- [I-D.ietf-core-dynlink] Shelby, Z., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Dynamic Resource Linking for Constrained RESTful Environments", draft-ietf-core-dynlink-08 (work in progress), March 2019.
- [I-D.ietf-core-resource-directory] Shelby, Z., Koster, M., Bormann, C., Stok, P., and C. Amsuess, "CoRE Resource Directory", draft-ietf-core-resource-directory-19 (work in progress), January 2019.
- [OIC-Core] "OIC Resource Type Specification v1.1.0", 2016, <<https://openconnectivity.org/resources/specifications>>.
- [OIC-SmartHome] "OIC Smart Home Device Specification v1.1.0", 2016, <<https://openconnectivity.org/resources/specifications>>.
- [OMA-TS-LWM2M] "Lightweight Machine to Machine Technical Specification", 2016, <<http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0>>.
- [oneM2MTS0008] "TS 0008 v1.3.2 CoAP Protocol Binding", 2016, <<http://www.onem2m.org/technical/published-documents>>.

- [oneM2MTS0023] "TS 0023 v2.0.0 Home Appliances Information Model and Mapping", 2016,
<<http://www.onem2m.org/technical/published-documents>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005,
<<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC6573] Amundsen, M., "The Item and Collection Link Relations", RFC 6573, DOI 10.17487/RFC6573, April 2012,
<<https://www.rfc-editor.org/info/rfc6573>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014,
<<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014,
<<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC7396] Hoffman, P. and J. Snell, "JSON Merge Patch", RFC 7396, DOI 10.17487/RFC7396, October 2014,
<<https://www.rfc-editor.org/info/rfc7396>>.
- [RFC8428] Jennings, C., Shelby, Z., Arkko, J., Keranen, A., and C. Bormann, "Sensor Measurement Lists (SenML)", RFC 8428, DOI 10.17487/RFC8428, August 2018,
<<https://www.rfc-editor.org/info/rfc8428>>.

Appendix A. Current Usage of Interfaces

Editor's note: This appendix will be removed. It is only included for information.

This appendix analyses the current landscape with regards the definition and use of collections and interfaces. This should be considered when considering the scope of this document.

A.1. Constrained RESTful Environments (CoRE) Link Format (IETF)

[RFC6690] assumes that different deployments or application domains will define the appropriate REST Interface Descriptions along with Resource Types to make discovery meaningful. It highlights that collections are often used for these interfaces.

Whilst 3.2/[RFC6690] defines a new Interface Description 'if' attribute the procedures around it are about the naming of the interface not what information should be included in the documentation about the interface.

A.2. Open Connectivity Foundation (OCF)

The OIC Core Specification [OIC-Core] most closely aligns with the work in this specification. It makes use of interface descriptions as per [RFC6690] and has registered several interface identifiers (<https://www.iana.org/assignments/core-parameters/core-parameters.xhtml#if-link-target-att-value>). These interface descriptors are similar to those defined in this specification. From a high level perspective:

```
links list:   OCF (oic.if.ll) -> IETF (core.ll)
               Note: it's called "link list" in the IETF.
linked batch: OCF (oic.if.b) -> IETF (core.lb)
read-only:    OCF (oic.if.r) -> IETF (core.rp)
read-write:   OCF (oic.if.rw) -> IETF (core.p)
actuator:     OCF (oic.if.a) -> IETF (core.a)
sensor:       OCF (oic.if.s) -> IETF (core.s)
batch:        No OCF equivalent -> IETF (core.b)
```

Some of the OCF interfaces make use of collections.

The OIC Core specification does not use the concept of function sets. It does however discuss the concept of profiles. The OCF defines two sets of documents. The core specification documents such as [OIC-Core] and vertical profile specification documents which provide specific information for specific applications. The OIC Smart Home Device Specification [OIC-SmartHome] is one such specification. It provides information on the resource model, discovery and data types.

Authors' Addresses

Zach Shelby
ARM
Kidekuja 2
Vuokatti 88600
FINLAND

Phone: +358407796297
Email: zach.shelby@arm.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Email: michael.koster@smarththings.com

Christian Groves
Australia

Email: cngroves.std@gmail.com

Jintao Zhu
Huawei
No.127 Jinye Road, Huawei Base, High-Tech Development District
Xi'an, Shaanxi Province
China

Email: jintao.zhu@huawei.com

Bilhanan Silverajan (editor)
Tampere University
Kalevantie 4
Tampere FI-33100
Finland

Email: bilhanan.silverajan@tuni.fi

CoRE 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 "parmname"). 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 ("parmname") 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 ("parmname") 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., 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>>.
- [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

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 Working Group
Internet-Draft
Updates: 7252 (if approved)
Intended status: Standards Track
Expires: September 7, 2019

G. Selander
J. Mattsson
F. Palombini
Ericsson AB
L. Seitz
RISE SICS
March 06, 2019

Object Security for Constrained RESTful Environments (OSCORE)
draft-ietf-core-object-security-16

Abstract

This document defines Object Security for Constrained RESTful Environments (OSCORE), a method for application-layer protection of the Constrained Application Protocol (CoAP), using CBOR Object Signing and Encryption (COSE). OSCORE provides end-to-end protection between endpoints communicating using CoAP or CoAP-mappable HTTP. OSCORE is designed for constrained nodes and networks supporting a range of proxy operations, including translation between different transport protocols.

Although being an optional functionality of CoAP, OSCORE alters CoAP options processing and IANA registration. Therefore, this document updates [RFC7252].

Status of This Memo

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

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

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

This Internet-Draft will expire on September 7, 2019.

Copyright Notice

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

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

Table of Contents

1. Introduction	4
1.1. Terminology	6
2. The OSCORE Option	7
3. The Security Context	7
3.1. Security Context Definition	8
3.2. Establishment of Security Context Parameters	10
3.3. Requirements on the Security Context Parameters	12
4. Protected Message Fields	13
4.1. CoAP Options	14
4.2. CoAP Header Fields and Payload	23
4.3. Signaling Messages	23
5. The COSE Object	24
5.1. ID Context and 'kid context'	25
5.2. AEAD Nonce	26
5.3. Plaintext	27
5.4. Additional Authenticated Data	28
6. OSCORE Header Compression	29
6.1. Encoding of the OSCORE Option Value	30
6.2. Encoding of the OSCORE Payload	31
6.3. Examples of Compressed COSE Objects	31
7. Message Binding, Sequence Numbers, Freshness, and Replay Protection	34
7.1. Message Binding	34
7.2. Sequence Numbers	34
7.3. Freshness	34
7.4. Replay Protection	35
7.5. Losing Part of the Context State	36
8. Processing	37
8.1. Protecting the Request	37
8.2. Verifying the Request	37
8.3. Protecting the Response	39

8.4. Verifying the Response	40
9. Web Linking	42
10. CoAP-to-CoAP Forwarding Proxy	42
11. HTTP Operations	43
11.1. The HTTP OSCORE Header Field	43
11.2. CoAP-to-HTTP Mapping	44
11.3. HTTP-to-CoAP Mapping	45
11.4. HTTP Endpoints	45
11.5. Example: HTTP Client and CoAP Server	46
11.6. Example: CoAP Client and HTTP Server	47
12. Security Considerations	48
12.1. End-to-end Protection	48
12.2. Security Context Establishment	49
12.3. Master Secret	49
12.4. Replay Protection	50
12.5. Client Aliveness	50
12.6. Cryptographic Considerations	50
12.7. Message Segmentation	51
12.8. Privacy Considerations	51
13. IANA Considerations	52
13.1. COSE Header Parameters Registry	52
13.2. CoAP Option Numbers Registry	53
13.3. CoAP Signaling Option Numbers Registry	54
13.4. Header Field Registrations	54
13.5. Media Type Registrations	54
13.6. CoAP Content-Formats Registry	56
13.7. OSCORE Flag Bits Registry	56
13.8. Expert Review Instructions	57
14. References	58
14.1. Normative References	58
14.2. Informative References	59
Appendix A. Scenario Examples	62
A.1. Secure Access to Sensor	62
A.2. Secure Subscribe to Sensor	63
Appendix B. Deployment Examples	64
B.1. Security Context Derived Once	64
B.2. Security Context Derived Multiple Times	66
Appendix C. Test Vectors	71
C.1. Test Vector 1: Key Derivation with Master Salt	72
C.2. Test Vector 2: Key Derivation without Master Salt	73
C.3. Test Vector 3: Key Derivation with ID Context	75
C.4. Test Vector 4: OSCORE Request, Client	76
C.5. Test Vector 5: OSCORE Request, Client	77
C.6. Test Vector 6: OSCORE Request, Client	79
C.7. Test Vector 7: OSCORE Response, Server	80
C.8. Test Vector 8: OSCORE Response with Partial IV, Server	81
Appendix D. Overview of Security Properties	82
D.1. Threat Model	82

D.2. Supporting Proxy Operations	83
D.3. Protected Message Fields	84
D.4. Uniqueness of (key, nonce)	85
D.5. Unprotected Message Fields	86
Appendix E. CDDL Summary	89
Acknowledgments	90
Authors' Addresses	90

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web transfer protocol, designed for constrained nodes and networks [RFC7228], and may be mapped from HTTP [RFC8075]. CoAP specifies the use of proxies for scalability and efficiency and references DTLS [RFC6347] for security. CoAP-to-CoAP, HTTP-to-CoAP, and CoAP-to-HTTP proxies require DTLS or TLS [RFC8446] to be terminated at the proxy. The proxy therefore not only has access to the data required for performing the intended proxy functionality, but is also able to eavesdrop on, or manipulate any part of, the message payload and metadata in transit between the endpoints. The proxy can also inject, delete, or reorder packets since they are no longer protected by (D)TLS.

This document defines the Object Security for Constrained RESTful Environments (OSCORE) security protocol, protecting CoAP and CoAP-mappable HTTP requests and responses end-to-end across intermediary nodes such as CoAP forward proxies and cross-protocol translators including HTTP-to-CoAP proxies [RFC8075]. In addition to the core CoAP features defined in [RFC7252], OSCORE supports the Observe [RFC7641], Block-wise [RFC7959], and No-Response [RFC7967] options, as well as the PATCH and FETCH methods [RFC8132]. An analysis of end-to-end security for CoAP messages through some types of intermediary nodes is performed in [I-D.hartke-core-e2e-security-reqs]. OSCORE essentially protects the RESTful interactions; the request method, the requested resource, the message payload, etc. (see Section 4). OSCORE protects neither the CoAP Messaging Layer nor the CoAP Token which may change between the endpoints, and those are therefore processed as defined in [RFC7252]. Additionally, since the message formats for CoAP over unreliable transport [RFC7252] and for CoAP over reliable transport [RFC8323] differ only in terms of CoAP Messaging Layer, OSCORE can be applied to both unreliable and reliable transports (see Figure 1).

OSCORE works in very constrained nodes and networks, thanks to its small message size and the restricted code and memory requirements in addition to what is required by CoAP. Examples of the use of OSCORE are given in Appendix A. OSCORE may be used over any underlying layer, such as e.g. UDP or TCP, and with non-IP transports (e.g.,

[I-D.bormann-6lo-coap-802-15-4e]). 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.

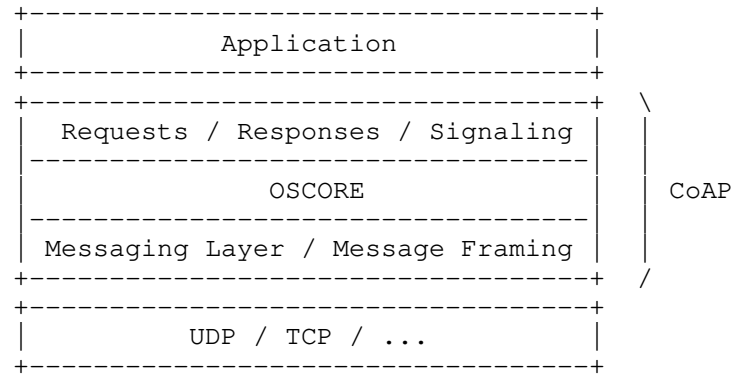


Figure 1: Abstract Layering of CoAP with OSCORE

OSCORE is designed to protect as much information as possible while still allowing CoAP proxy operations (Section 10). It works with existing CoAP-to-CoAP forward proxies [RFC7252], but an OSCORE-aware proxy will be more efficient. HTTP-to-CoAP proxies [RFC8075] and CoAP-to-HTTP proxies can also be used with OSCORE, as specified in Section 11. OSCORE may be used together with TLS or DTLS over one or more hops in the end-to-end path, e.g. transported with HTTPS in one hop and with plain CoAP in another hop. The use of OSCORE does not affect the URI scheme and OSCORE can therefore be used with any URI scheme defined for CoAP or HTTP. The application decides the conditions for which OSCORE is required.

OSCORE uses pre-shared keys which may have been established out-of-band or with a key establishment protocol (see Section 3.2). The technical solution builds on CBOR Object Signing and Encryption (COSE) [RFC8152], providing end-to-end encryption, integrity, replay protection, and binding of response to request. A compressed version of COSE is used, as specified in Section 6. The use of OSCORE is signaled in CoAP with a new option (Section 2), and in HTTP with a new header field (Section 11.1) and content type (Section 13.5). The solution transforms a CoAP/HTTP message into an "OSCORE message" before sending, and vice versa after receiving. The OSCORE message is a CoAP/HTTP message related to the original message in the following way: the original CoAP/HTTP message is translated to CoAP (if not already in CoAP) and protected in a COSE object. The encrypted message fields of this COSE object are transported in the CoAP payload/HTTP body of the OSCORE message, and the OSCORE option/

header field is included in the message. A sketch of an exchange of OSCORE messages, in the case of the original message being CoAP, is provided in Figure 2. The use of OSCORE with HTTP is detailed in Section 11.

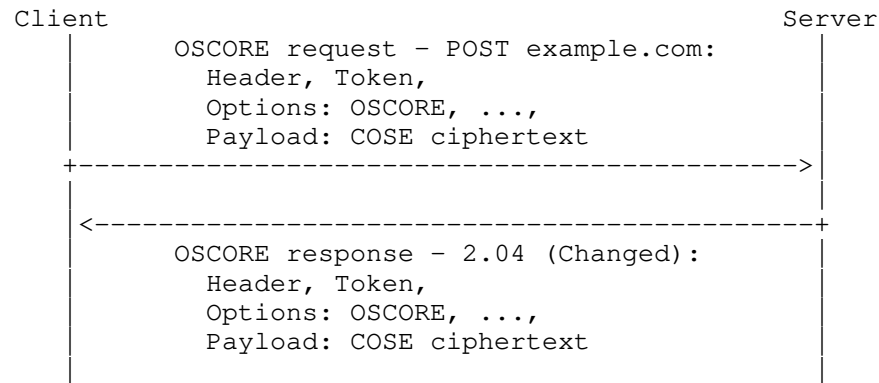


Figure 2: Sketch of CoAP with OSCORE

An implementation supporting this specification MAY implement only the client part, MAY implement only the server part, or MAY implement only one of the proxy parts.

1.1. Terminology

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

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252], Observe [RFC7641], Block-wise [RFC7959], COSE [RFC8152], CBOR [RFC7049], CDDL [I-D.ietf-cbor-cddl] as summarized in Appendix E, and constrained environments [RFC7228].

The term "hop" is used to denote a particular leg in the end-to-end path. The concept "hop-by-hop" (as in "hop-by-hop encryption" or "hop-by-hop fragmentation") opposed to "end-to-end", is used in this document to indicate that the messages are processed accordingly in the intermediaries, rather than just forwarded to the next node.

The term "stop processing" is used throughout the document to denote that the message is not passed up to the CoAP Request/Response Layer (see Figure 1).

The terms Common/Sender/Recipient Context, Master Secret/Salt, Sender ID/Key, Recipient ID/Key, ID Context, and Common IV are defined in Section 3.1.

2. The OSCORE Option

The OSCORE option defined in this section (see Figure 3, which extends Table 4: Options of [RFC7252]) indicates that the CoAP message is an OSCORE message and that it contains a compressed COSE object (see Sections 5 and 6). The OSCORE option is critical, safe to forward, part of the cache key, and not repeatable.

No.	C	U	N	R	Name	Format	Length	Default
TBD1	x				OSCORE	(*)	0-255	(none)

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable
 (*) See below.

Figure 3: The OSCORE Option

The OSCORE option includes the OSCORE flag bits (Section 6), the Sender Sequence Number, the Sender ID, and the ID Context when these fields are present (Section 3). The detailed format and length is specified in Section 6. If the OSCORE flag bits are all zero (0x00) the Option value SHALL be empty (Option Length = 0). An endpoint receiving a CoAP message without payload, that also contains an OSCORE option SHALL treat it as malformed and reject it.

A successful response to a request with the OSCORE option SHALL contain the OSCORE option. Whether error responses contain the OSCORE option depends on the error type (see Section 8).

For CoAP proxy operations, see Section 10.

3. The Security Context

OSCORE requires that client and server establish a shared security context used to process the COSE objects. OSCORE uses COSE with an Authenticated Encryption with Additional Data (AEAD, [RFC5116]) algorithm for protecting message data between a client and a server. In this section, we define the security context and how it is derived in client and server based on a shared secret and a key derivation function.

3.1. Security Context Definition

The security context is the set of information elements necessary to carry out the cryptographic operations in OSCORE. For each endpoint, the security context is composed of a "Common Context", a "Sender Context", and a "Recipient Context".

The endpoints protect messages to send using the Sender Context and verify messages received using the Recipient Context, both contexts being derived from the Common Context and other data. Clients and servers need to be able to retrieve the correct security context to use.

An endpoint uses its Sender ID (SID) to derive its Sender Context, and the other endpoint uses the same ID, now called Recipient ID (RID), to derive its Recipient Context. In communication between two endpoints, the Sender Context of one endpoint matches the Recipient Context of the other endpoint, and vice versa. Thus, the two security contexts identified by the same IDs in the two endpoints are not the same, but they are partly mirrored. Retrieval and use of the security context are shown in Figure 4.

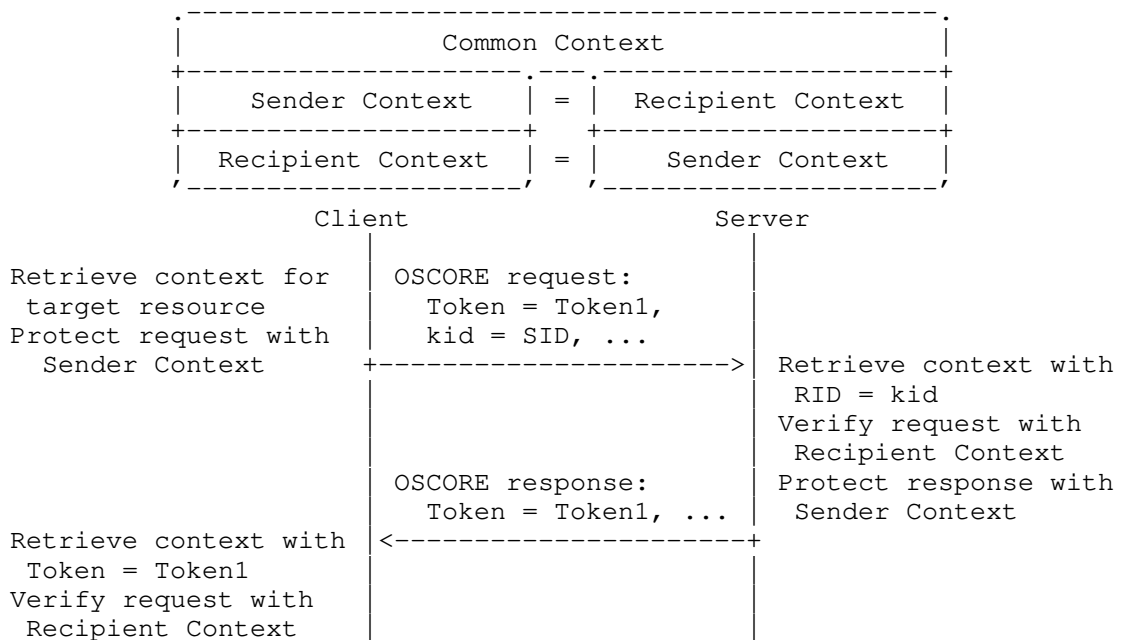


Figure 4: Retrieval and Use of the Security Context

The Common Context contains the following parameters:

- o AEAD Algorithm. The COSE AEAD algorithm to use for encryption.
- o HKDF Algorithm. An HMAC-based key derivation function HKDF [RFC5869] used to derive Sender Key, Recipient Key, and Common IV.
- o Master Secret. Variable length, random byte string (see Section 12.3) used to derive AEAD keys and Common IV.
- o Master Salt. Optional variable length byte string containing the salt used to derive AEAD keys and Common IV.
- o ID Context. Optional variable length byte string providing additional information to identify the Common Context and to derive AEAD keys and Common IV. The use of ID Context is described in Section 5.1.
- o Common IV. Byte string derived from Master Secret, Master Salt, and ID Context. Used to generate the AEAD Nonce (see Section 5.2). Same length as the nonce of the AEAD Algorithm.

The Sender Context contains the following parameters:

- o Sender ID. Byte string used to identify the Sender Context, to derive AEAD keys and Common IV, and to assure unique AEAD nonces. Maximum length is determined by the AEAD Algorithm.
- o Sender Key. Byte string containing the symmetric AEAD key to protect messages to send. Derived from Common Context and Sender ID. Length is determined by the AEAD Algorithm.
- o Sender Sequence Number. Non-negative integer used by the sender to enumerate requests and certain responses, e.g. Observe notifications. Used as 'Partial IV' [RFC8152] to generate unique AEAD nonces. Maximum value is determined by the AEAD Algorithm. Initialization is described in Section 3.2.2.

The Recipient Context contains the following parameters:

- o Recipient ID. Byte string used to identify the Recipient Context, to derive AEAD keys and Common IV, and to assure unique AEAD nonces. Maximum length is determined by the AEAD Algorithm.
- o Recipient Key. Byte string containing the symmetric AEAD key to verify messages received. Derived from Common Context and Recipient ID. Length is determined by the AEAD Algorithm.

- o Replay Window (Server only). The replay window to verify requests received. Replay protection is described in Section 7.4 and Section 3.2.2.

All parameters except Sender Sequence Number and Replay Window are immutable once the security context is established. An endpoint may free up memory by not storing the Common IV, Sender Key, and Recipient Key, deriving them when needed. Alternatively, an endpoint may free up memory by not storing the Master Secret and Master Salt after the other parameters have been derived.

Endpoints MAY operate as both client and server and use the same security context for those roles. Independent of being client or server, the endpoint protects messages to send using its Sender Context, and verifies messages received using its Recipient Context. The endpoints MUST NOT change the Sender/Recipient ID when changing roles. In other words, changing the roles does not change the set of AEAD keys to be used.

3.2. Establishment of Security Context Parameters

Each endpoint derives the parameters in the security context from a small set of input parameters. The following input parameters SHALL be pre-established:

- o Master Secret
- o Sender ID
- o Recipient ID

The following input parameters MAY be pre-established. In case any of these parameters is not pre-established, the default value indicated below is used:

- o AEAD Algorithm
 - * Default is AES-CCM-16-64-128 (COSE algorithm encoding: 10)
- o Master Salt
 - * Default is the empty byte string
- o HKDF Algorithm
 - * Default is HKDF SHA-256
- o Replay Window

- * Default is DTLS-type replay protection with a window size of 32 [RFC6347]

All input parameters need to be known to and agreed on by both endpoints, but the replay window may be different in the two endpoints. The way the input parameters are pre-established, is application specific. Considerations of security context establishment are given in Section 12.2 and examples of deploying OSCORE in Appendix B.

3.2.1. Derivation of Sender Key, Recipient Key, and Common IV

The HKDF MUST be one of the HMAC-based HKDF [RFC5869] algorithms defined for COSE [RFC8152]. HKDF SHA-256 is mandatory to implement. The security context parameters Sender Key, Recipient Key, and Common IV SHALL be derived from the input parameters using the HKDF, which consists of the composition of the HKDF-Extract and HKDF-Expand steps [RFC5869]:

```
output parameter = HKDF(salt, IKM, info, L)
```

where:

- o salt is the Master Salt as defined above
- o IKM is the Master Secret as defined above
- o info is the serialization of a CBOR array consisting of (the notation follows Appendix E):

```
info = [  
  id : bstr,  
  id_context : bstr / nil,  
  alg_aead : int / tstr,  
  type : tstr,  
  L : uint,  
]
```

where:

- o id is the Sender ID or Recipient ID when deriving Sender Key and Recipient Key, respectively, and the empty byte string when deriving the Common IV.
- o id_context is the ID Context, or nil if ID Context is not provided.
- o alg_aead is the AEAD Algorithm, encoded as defined in [RFC8152].

- o type is "Key" or "IV". The label is an ASCII string, and does not include a trailing NUL byte.
- o L is the size of the key/nonce for the AEAD algorithm used, in bytes.

For example, if the algorithm AES-CCM-16-64-128 (see Section 10.2 in [RFC8152]) is used, the integer value for alg_aead is 10, the value for L is 16 for keys and 13 for the Common IV. Assuming use of the default algorithms HKDF SHA-256 and AES-CCM-16-64-128, the extract phase of HKDF produces a pseudorandom key (PRK) as follows:

PRK = HMAC-SHA-256(Master Salt, Master Secret)

and as L is smaller than the hash function output size, the expand phase of HKDF consists of a single HMAC invocation, and the Sender Key, Recipient Key, and Common IV are therefore the first 16 or 13 bytes of

output parameter = HMAC-SHA-256(PRK, info || 0x01)

where different info are used for each derived parameter and where || denotes byte string concatenation.

Note that [RFC5869] specifies that if the salt is not provided, it is set to a string of zeros. For implementation purposes, not providing the salt is the same as setting the salt to the empty byte string. OSCORE sets the salt default value to empty byte string, which is converted to a string of zeroes (see Section 2.2 of [RFC5869]).

3.2.2. Initial Sequence Numbers and Replay Window

The Sender Sequence Number is initialized to 0.

The supported types of replay protection and replay window length is application specific and depends on how OSCORE is transported, see Section 7.4. The default is DTLS-type replay protection with a window size of 32 initiated as described in Section 4.1.2.6 of [RFC6347].

3.3. Requirements on the Security Context Parameters

To ensure unique Sender Keys, the quartet (Master Secret, Master Salt, ID Context, Sender ID) MUST be unique, i.e. the pair (ID Context, Sender ID) SHALL be unique in the set of all security contexts using the same Master Secret and Master Salt. This means that Sender ID SHALL be unique in the set of all security contexts

using the same Master Secret, Master Salt, and ID Context; such a requirement guarantees unique (key, nonce) pairs for the AEAD.

Different methods can be used to assign Sender IDs: a protocol that allows the parties to negotiate locally unique identifiers, a trusted third party (e.g., [I-D.ietf-ace-oauth-authz]), or the identifiers can be assigned out-of-band. The Sender IDs can be very short (note that the empty string is a legitimate value). The maximum length of Sender ID in bytes equals the length of AEAD nonce minus 6, see Section 5.2. For AES-CCM-16-64-128 the maximum length of Sender ID is 7 bytes.

To simplify retrieval of the right Recipient Context, the Recipient ID SHOULD be unique in the sets of all Recipient Contexts used by an endpoint. If an endpoint has the same Recipient ID with different Recipient Contexts, i.e. the Recipient Contexts are derived from different Common Contexts, then the endpoint may need to try multiple times before verifying the right security context associated to the Recipient ID.

The ID Context is used to distinguish between security contexts. The methods used for assigning Sender ID can also be used for assigning the ID Context. Additionally, the ID Context can be used to introduce randomness into new Sender and Recipient Contexts (see Appendix B.2). ID Context can be arbitrarily long.

4. Protected Message Fields

OSCORE transforms a CoAP message (which may have been generated from an HTTP message) into an OSCORE message, and vice versa. OSCORE protects as much of the original message as possible while still allowing certain proxy operations (see Sections 10 and 11). This section defines how OSCORE protects the message fields and transfers them end-to-end between client and server (in any direction).

The remainder of this section and later sections focus on the behavior in terms of CoAP messages. If HTTP is used for a particular hop in the end-to-end path, then this section applies to the conceptual CoAP message that is mappable to/from the original HTTP message as discussed in Section 11. That is, an HTTP message is conceptually transformed to a CoAP message and then to an OSCORE message, and similarly in the reverse direction. An actual implementation might translate directly from HTTP to OSCORE without the intervening CoAP representation.

Protection of Signaling messages (Section 5 of [RFC8323]) is specified in Section 4.3. The other parts of this section target Request/Response messages.

Message fields of the CoAP message may be protected end-to-end between CoAP client and CoAP server in different ways:

- o Class E: encrypted and integrity protected,
- o Class I: integrity protected only, or
- o Class U: unprotected.

The sending endpoint SHALL transfer Class E message fields in the ciphertext of the COSE object in the OSCORE message. The sending endpoint SHALL include Class I message fields in the Additional Authenticated Data (AAD) of the AEAD algorithm, allowing the receiving endpoint to detect if the value has changed in transfer. Class U message fields SHALL NOT be protected in transfer. Class I and Class U message field values are transferred in the header or options part of the OSCORE message, which is visible to proxies.

Message fields not visible to proxies, i.e., transported in the ciphertext of the COSE object, are called "Inner" (Class E). Message fields transferred in the header or options part of the OSCORE message, which is visible to proxies, are called "Outer" (Class I or U). There are currently no Class I options defined.

An OSCORE message may contain both an Inner and an Outer instance of a certain CoAP message field. Inner message fields are intended for the receiving endpoint, whereas Outer message fields are used to enable proxy operations.

4.1. CoAP Options

A summary of how options are protected is shown in Figure 5. Note that some options may have both Inner and Outer message fields which are protected accordingly. Certain options require special processing as is described in Section 4.1.3.

Options that are unknown or for which OSCORE processing is not defined SHALL be processed as class E (and no special processing). Specifications of new CoAP options SHOULD define how they are processed with OSCORE. A new COAP option SHOULD be of class E unless it requires proxy processing. If a new CoAP option is of class U, the potential issues with the option being unprotected SHOULD be documented (see Appendix D.5).

4.1.1.1. Inner Options

Inner option message fields (class E) are used to communicate directly with the other endpoint.

The sending endpoint SHALL write the Inner option message fields present in the original CoAP message into the plaintext of the COSE object (Section 5.3), and then remove the Inner option message fields from the OSCORE message.

The processing of Inner option message fields by the receiving endpoint is specified in Sections 8.2 and 8.4.

No.	Name	E	U
1	If-Match	x	
3	Uri-Host		x
4	ETag	x	
5	If-None-Match	x	
6	Observe	x	x
7	Uri-Port		x
8	Location-Path	x	
TBD1	OSCORE		x
11	Uri-Path	x	
12	Content-Format	x	
14	Max-Age	x	x
15	Uri-Query	x	
17	Accept	x	
20	Location-Query	x	
23	Block2	x	x
27	Block1	x	x
28	Size2	x	x
35	Proxy-Uri		x
39	Proxy-Scheme		x
60	Size1	x	x
258	No-Response	x	x

E = Encrypt and Integrity Protect (Inner)
U = Unprotected (Outer)

Figure 5: Protection of CoAP Options

4.1.2. Outer Options

Outer option message fields (Class U or I) are used to support proxy operations, see Appendix D.2.

The sending endpoint SHALL include the Outer option message field present in the original message in the options part of the OSCORE message. All Outer option message fields, including the OSCORE option, SHALL be encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of Outer option message field.

The processing of Outer options by the receiving endpoint is specified in Sections 8.2 and 8.4.

A procedure for integrity-protection-only of Class I option message fields is specified in Section 5.4. Specifications that introduce repeatable Class I options MUST specify that proxies MUST NOT change the order of the instances of such an option in the CoAP message.

Note: There are currently no Class I option message fields defined.

4.1.3. Special Options

Some options require special processing as specified in this section.

4.1.3.1. Max-Age

An Inner Max-Age message field is used to indicate the maximum time a response may be cached by the client (as defined in [RFC7252]), end-to-end from the server to the client, taking into account that the option is not accessible to proxies. The Inner Max-Age SHALL be processed by OSCORE as a normal Inner option, specified in Section 4.1.1.

An Outer Max-Age message field is used to avoid unnecessary caching of error responses caused by OSCORE processing at OSCORE-unaware intermediary nodes. A server MAY set a Class U Max-Age message field with value zero to such error responses, described in Sections 7.4, 8.2, and 8.4, since these error responses are cacheable, but subsequent OSCORE requests would never create a hit in the intermediary caching it. Setting the Outer Max-Age to zero relieves the intermediary from uselessly caching responses. Successful OSCORE responses do not need to include an Outer Max-Age option since the responses appear to the OSCORE-unaware intermediary as 2.04 (Changed) responses, which are non-cacheable (see Section 4.2).

The Outer Max-Age message field is processed according to Section 4.1.2.

4.1.3.2. Uri-Host and Uri-Port

When the Uri-Host and Uri-Port are set to their default values (see Section 5.10.1 [RFC7252]), they are omitted from the message (Section 5.4.4 of [RFC7252]), which is favorable both for overhead and privacy.

In order to support forward proxy operations, Proxy-Scheme, Uri-Host, and Uri-Port need to be Class U. For the use of Proxy-Uri, see Section 4.1.3.3.

Manipulation of unprotected message fields (including Uri-Host, Uri-Port, destination IP/port or request scheme) MUST NOT lead to an OSCORE message becoming verified by an unintended server. Different servers SHALL have different security contexts.

4.1.3.3. Proxy-Uri

When Proxy-Uri is present, the client SHALL first decompose the Proxy-Uri value of the original CoAP message into the Proxy-Scheme, Uri-Host, Uri-Port, Uri-Path, and Uri-Query options according to Section 6.4 of [RFC7252].

Uri-Path and Uri-Query are class E options and SHALL be protected and processed as Inner options (Section 4.1.1).

The Proxy-Uri option of the OSCORE message SHALL be set to the composition of Proxy-Scheme, Uri-Host, and Uri-Port options as specified in Section 6.5 of [RFC7252], and processed as an Outer option of Class U (Section 4.1.2).

Note that replacing the Proxy-Uri value with the Proxy-Scheme and Uri-* options works by design for all CoAP URIs (see Section 6 of [RFC7252]). OSCORE-aware HTTP servers should not use the userinfo component of the HTTP URI (as defined in Section 3.2.1 of [RFC3986]), so that this type of replacement is possible in the presence of CoAP-to-HTTP proxies (see Section 11.2). In future specifications of cross-protocol proxying behavior using different URI structures, it is expected that the authors will create Uri-* options that allow decomposing the Proxy-Uri, and specifying the OSCORE processing.

An example of how Proxy-Uri is processed is given here. Assume that the original CoAP message contains:

```
o Proxy-Uri = "coap://example.com/resource?q=1"
```

During OSCORE processing, Proxy-Uri is split into:

- o Proxy-Scheme = "coap"
- o Uri-Host = "example.com"
- o Uri-Port = "5683"
- o Uri-Path = "resource"
- o Uri-Query = "q=1"

Uri-Path and Uri-Query follow the processing defined in Section 4.1.1, and are thus encrypted and transported in the COSE object:

- o Uri-Path = "resource"
- o Uri-Query = "q=1"

The remaining options are composed into the Proxy-Uri included in the options part of the OSCORE message, which has value:

- o Proxy-Uri = "coap://example.com"

See Sections 6.1 and 12.6 of [RFC7252] for more details.

4.1.3.4. The Block Options

Block-wise [RFC7959] is an optional feature. An implementation MAY support [RFC7252] and the OSCORE option without supporting block-wise transfers. The Block options (Block1, Block2, Size1, Size2), when Inner message fields, provide secure message segmentation such that each segment can be verified. The Block options, when Outer message fields, enables hop-by-hop fragmentation of the OSCORE message. Inner and Outer block processing may have different performance properties depending on the underlying transport. The end-to-end integrity of the message can be verified both in case of Inner and Outer Block-wise transfers provided all blocks are received.

4.1.3.4.1. Inner Block Options

The sending CoAP endpoint MAY fragment a CoAP message as defined in [RFC7959] before the message is processed by OSCORE. In this case the Block options SHALL be processed by OSCORE as normal Inner options (Section 4.1.1). The receiving CoAP endpoint SHALL process the OSCORE message before processing Block-wise as defined in [RFC7959].

4.1.3.4.2. Outer Block Options

Proxies MAY fragment an OSCORE message using [RFC7959], by introducing Block option message fields that are Outer (Section 4.1.2). Note that the Outer Block options are neither encrypted nor integrity protected. As a consequence, a proxy can maliciously inject block fragments indefinitely, since the receiving endpoint needs to receive the last block (see [RFC7959]) to be able to compose the OSCORE message and verify its integrity. Therefore, applications supporting OSCORE and [RFC7959] MUST specify a security policy defining a maximum unfragmented message size (MAX_UNFRAGMENTED_SIZE) considering the maximum size of message which can be handled by the endpoints. Messages exceeding this size SHOULD be fragmented by the sending endpoint using Inner Block options (Section 4.1.3.4.1).

An endpoint receiving an OSCORE message with an Outer Block option SHALL first process this option according to [RFC7959], until all blocks of the OSCORE message have been received, or the cumulated message size of the blocks exceeds MAX_UNFRAGMENTED_SIZE. In the former case, the processing of the OSCORE message continues as defined in this document. In the latter case the message SHALL be discarded.

Because of encryption of Uri-Path and Uri-Query, messages to the same server may, from the point of view of a proxy, look like they also target the same resource. A proxy SHOULD mitigate a potential mix-up of blocks from concurrent requests to the same server, for example using the Request-Tag processing specified in Section 3.3.2 of [I-D.ietf-core-echo-request-tag].

4.1.3.5. Observe

Observe [RFC7641] is an optional feature. An implementation MAY support [RFC7252] and the OSCORE option without supporting [RFC7641], in which case the Observe related processing can be omitted.

The support for Observe [RFC7641] with OSCORE targets the requirements on forwarding of Section 2.2.1 of [I-D.hartke-core-e2e-security-reqs], i.e. that observations go through intermediary nodes, as illustrated in Figure 8 of [RFC7641].

Inner Observe SHALL be used to protect the value of the Observe option between the endpoints. Outer Observe SHALL be used to support forwarding by intermediary nodes.

The server SHALL include a new Partial IV (see Section 5) in responses (with or without the Observe option) to Observe

registrations, except for the first response where Partial IV MAY be omitted.

For cancellations, Section 3.6 of [RFC7641] specifies that all options MUST be identical to those in the registration request except for Observe and the set of ETag Options. For OSCORE messages, this matching is to be done to the options in the decrypted message.

[RFC7252] does not specify how the server should act upon receiving the same Token in different requests. When using OSCORE, the server SHOULD NOT remove an active observation just because it receives a request with the same Token.

Since POST with Observe is not defined, for messages with Observe, the Outer Code MUST be set to 0.05 (FETCH) for requests and to 2.05 (Content) for responses (see Section 4.2).

4.1.3.5.1. Registrations and Cancellations

The Inner and Outer Observe in the request MUST contain the Observe value of the original CoAP request; 0 (registration) or 1 (cancellation).

Every time a client issues a new Observe request, a new Partial IV MUST be used (see Section 5), and so the payload and OSCORE option are changed. The server uses the Partial IV of the new request as the 'request_piv' of all associated notifications (see Section 5.4).

Intermediaries are not assumed to have access to the OSCORE security context used by the endpoints, and thus cannot make requests or transform responses with the OSCORE option which verify at the receiving endpoint as coming from the other endpoint. This has the following consequences and limitations for Observe operations.

- o An intermediary node removing the Outer Observe 0 does not change the registration request to a request without Observe (see Section 2 of [RFC7641]). Instead other means for cancellation may be used as described in Section 3.6 of [RFC7641].
- o An intermediary node is not able to transform a normal response into an OSCORE protected Observe notification (see figure 7 of [RFC7641]) which verifies as coming from the server.
- o An intermediary node is not able to initiate an OSCORE protected Observe registration (Observe with value 0) which verifies as coming from the client. An OSCORE-aware intermediary SHALL NOT initiate registrations of observations (see Section 10). If an OSCORE-unaware proxy re-sends an old registration message from a

client this will trigger the replay protection mechanism in the server. To prevent this from resulting in the OSCORE-unaware proxy to cancel of the registration, a server MAY respond to a replayed registration request with a replay of a cached notification. Alternatively, the server MAY send a new notification.

- o An intermediary node is not able to initiate an OSCORE protected Observe cancellation (Observe with value 1) which verifies as coming from the client. An application MAY decide to allow intermediaries to cancel Observe registrations, e.g. to send Observe with value 1 (see Section 3.6 of [RFC7641]), but that can also be done with other methods, e.g. reusing the Token in a different request or sending a RST message. This is out of scope for this specification.

4.1.3.5.2. Notifications

If the server accepts an Observe registration, a Partial IV MUST be included in all notifications (both successful and error), except for the first one where Partial IV MAY be omitted. To protect against replay, the client SHALL maintain a Notification Number for each Observation it registers. The Notification Number is a non-negative integer containing the largest Partial IV of the received notifications for the associated Observe registration. Further details of replay protection of notifications are specified in Section 7.4.1.

For notifications, the Inner Observe value MUST be empty (see Section 3.2 of [RFC7252]). The Outer Observe in a notification is needed for intermediary nodes to allow multiple responses to one request, and may be set to the value of Observe in the original CoAP message. The client performs ordering of notifications and replay protection by comparing their Partial IVs and SHALL ignore the outer Observe value.

If the client receives a response to an Observe request without an Inner Observe option, then it verifies the response as a non-Observe response, as specified in Section 8.4. If the client receives a response to a non-Observe request with an Inner Observe option, then it stops processing the message, as specified in Section 8.4.

A client MUST consider the notification with the highest Partial IV as the freshest, regardless of the order of arrival. In order to support existing Observe implementations the OSCORE client implementation MAY set the Observe value to the three least significant bytes of the Partial IV. Implementations need to make

sure that the notification without Partial IV is considered the oldest.

4.1.3.6. No-Response

No-Response [RFC7967] is an optional feature used by the client to communicate its disinterest in certain classes of responses to a particular request. An implementation MAY support [RFC7252] and the OSCORE option without supporting [RFC7967].

If used, No-Response MUST be Inner. The Inner No-Response SHALL be processed by OSCORE as specified in Section 4.1.1. The Outer option SHOULD NOT be present. The server SHALL ignore the Outer No-Response option. The client MAY set the Outer No-Response value to 26 ('suppress all known codes') if the Inner value is set to 26. The client MUST be prepared to receive and discard 5.04 (Gateway Timeout) error messages from intermediaries potentially resulting from destination time out due to no response.

4.1.3.7. OSCORE

The OSCORE option is only defined to be present in OSCORE messages, as an indication that OSCORE processing have been performed. The content in the OSCORE option is neither encrypted nor integrity protected as a whole but some part of the content of this option is protected (see Section 5.4). Nested use of OSCORE is not supported: If OSCORE processing detects an OSCORE option in the original CoAP message, then processing SHALL be stopped.

Field	E	U
Version (UDP)		x
Type (UDP)		x
Length (TCP)		x
Token Length		x
Code	x	
Message ID (UDP)		x
Token		x
Payload	x	

E = Encrypt and Integrity Protect (Inner)
U = Unprotected (Outer)

Figure 6: Protection of CoAP Header Fields and Payload

4.2. CoAP Header Fields and Payload

A summary of how the CoAP header fields and payload are protected is shown in Figure 6, including fields specific to CoAP over UDP and CoAP over TCP (marked accordingly in the table).

Most CoAP Header fields (i.e. the message fields in the fixed 4-byte header) are required to be read and/or changed by CoAP proxies and thus cannot in general be protected end-to-end between the endpoints. As mentioned in Section 1, OSCORE protects the CoAP Request/Response Layer only, and not the Messaging Layer (Section 2 of [RFC7252]), so fields such as Type and Message ID are not protected with OSCORE.

The CoAP Header field Code is protected by OSCORE. Code SHALL be encrypted and integrity protected (Class E) to prevent an intermediary from eavesdropping on or manipulating the Code (e.g., changing from GET to DELETE).

The sending endpoint SHALL write the Code of the original CoAP message into the plaintext of the COSE object (see Section 5.3). After that, the sending endpoint writes an Outer Code to the OSCORE message. With one exception (see Section 4.1.3.5) the Outer Code SHALL be set to 0.02 (POST) for requests and to 2.04 (Changed) for responses. The receiving endpoint SHALL discard the Outer Code in the OSCORE message and write the Code of the COSE object plaintext (Section 5.3) into the decrypted CoAP message.

The other currently defined CoAP Header fields are Unprotected (Class U). The sending endpoint SHALL write all other header fields of the original message into the header of the OSCORE message. The receiving endpoint SHALL write the header fields from the received OSCORE message into the header of the decrypted CoAP message.

The CoAP Payload, if present in the original CoAP message, SHALL be encrypted and integrity protected and is thus an Inner message field. The sending endpoint writes the payload of the original CoAP message into the plaintext (Section 5.3) input to the COSE object. The receiving endpoint verifies and decrypts the COSE object, and recreates the payload of the original CoAP message.

4.3. Signaling Messages

Signaling messages (CoAP Code 7.00–7.31) were introduced to exchange information related to an underlying transport connection in the specific case of CoAP over reliable transports [RFC8323].

OSCORE MAY be used to protect Signaling if the endpoints for OSCORE coincide with the endpoints for the signaling message. If OSCORE is used to protect Signaling then:

- o To comply with [RFC8323], an initial empty CSM message SHALL be sent. The subsequent signaling message SHALL be protected.
- o Signaling messages SHALL be protected as CoAP Request messages, except in the case the Signaling message is a response to a previous Signaling message, in which case it SHALL be protected as a CoAP Response message. For example, 7.02 (Ping) is protected as a CoAP Request and 7.03 (Pong) as a CoAP response.
- o The Outer Code for Signaling messages SHALL be set to 0.02 (POST), unless it is a response to a previous Signaling message, in which case it SHALL be set to 2.04 (Changed).
- o All Signaling options, except the OSCORE option, SHALL be Inner (Class E).

NOTE: Option numbers for Signaling messages are specific to the CoAP Code (see Section 5.2 of [RFC8323]).

If OSCORE is not used to protect Signaling, Signaling messages SHALL be unaltered by OSCORE.

5. The COSE Object

This section defines how to use COSE [RFC8152] to wrap and protect data in the original message. OSCORE uses the untagged COSE_Encrypt0 structure with an Authenticated Encryption with Additional Data (AEAD) algorithm. The AEAD key lengths, AEAD nonce length, and maximum Sender Sequence Number are algorithm dependent.

The AEAD algorithm AES-CCM-16-64-128 defined in Section 10.2 of [RFC8152] is mandatory to implement. For AES-CCM-16-64-128 the length of Sender Key and Recipient Key is 128 bits, the length of AEAD nonce and Common IV is 13 bytes. The maximum Sender Sequence Number is specified in Section 12.

As specified in [RFC5116], plaintext denotes the data that is to be encrypted and integrity protected, and Additional Authenticated Data (AAD) denotes the data that is to be integrity protected only.

The COSE Object SHALL be a COSE_Encrypt0 object with fields defined as follows

- o The 'protected' field is empty.

- o The 'unprotected' field includes:
 - * The 'Partial IV' parameter. The value is set to the Sender Sequence Number. All leading bytes of value zero SHALL be removed when encoding the Partial IV, except in the case of Partial IV of value 0 which is encoded to the byte string 0x00. This parameter SHALL be present in requests. The Partial IV SHALL be present in responses to Observe registrations (see Section 4.1.3.5.1), otherwise the Partial IV will not typically be present in responses (for one exception, see Appendix B.1.2).
 - * The 'kid' parameter. The value is set to the Sender ID. This parameter SHALL be present in requests and will not typically be present in responses. An example where the Sender ID is included in a response is the extension of OSCORE to group communication [I-D.ietf-core-oscore-groupcomm].
 - * Optionally, a 'kid context' parameter (see Section 5.1). This parameter MAY be present in requests, and if so, MUST contain an ID Context (see Section 3.1). This parameter SHOULD NOT be present in responses: an example of how 'kid context' can be used in responses is given in Appendix B.2. If 'kid context' is present in the request, then the server SHALL use a security context with that ID Context when verifying the request.
- o The 'ciphertext' field is computed from the secret key (Sender Key or Recipient Key), AEAD nonce (see Section 5.2), plaintext (see Section 5.3), and the Additional Authenticated Data (AAD) (see Section 5.4) following Section 5.2 of [RFC8152].

The encryption process is described in Section 5.3 of [RFC8152].

5.1. ID Context and 'kid context'

For certain use cases, e.g. deployments where the same Sender ID is used with multiple contexts, it is possible (and sometimes necessary, see Section 3.3) for the client to use an ID Context to distinguish the security contexts (see Section 3.1). For example:

- o If the client has a unique identifier in some namespace then that identifier can be used as ID Context.
- o The ID Context may be used to add randomness into new Sender and Recipient Contexts, see Appendix B.2.

- o In case of group communication [I-D.ietf-core-oscore-groupcomm], a group identifier is used as ID Context to enable different security contexts for a server belonging to multiple groups.

The Sender ID and ID Context are used to establish the necessary input parameters and in the derivation of the security context (see Section 3.2).

Whereas the 'kid' parameter is used to transport the Sender ID, the new COSE header parameter 'kid context' is used to transport the ID Context in requests, see Figure 7.

name	label	value type	value registry	description
kid context	TBD2	bstr		Identifies the context for kid

Figure 7: Common Header Parameter 'kid context' for the COSE object

If ID Context is non-empty and the client sends a request without 'kid context' which results in an error indicating that the server could not find the security context, then the client could include the ID Context in the 'kid context' when making another request. Note that since the error is unprotected it may have been spoofed and the real response blocked by an on-path attacker.

5.2. AEAD Nonce

The high level design of the AEAD nonce follows Section 4.4 of [I-D.mcgregw-iv-gen], here follows the detailed construction (see Figure 8):

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

Note that in this specification only AEAD algorithms that use nonces equal or greater than 7 bytes are supported. The nonce construction with S, ID_PIV, and PIV together with endpoint unique IDs and

encryption keys makes it easy to verify that the nonces used with a specific key will be unique, see Appendix D.4.

If the Partial IV is not present in a response, the nonce from the request is used. For responses that are not notifications (i.e. when there is a single response to a request), the request and the response should typically use the same nonce to reduce message overhead. Both alternatives provide all the required security properties, see Section 7.4 and Appendix D.4. The only non-Observe scenario where a Partial IV must be included in a response is when the server is unable to perform replay protection, see Appendix B.1.2. For processing instructions see Section 8.

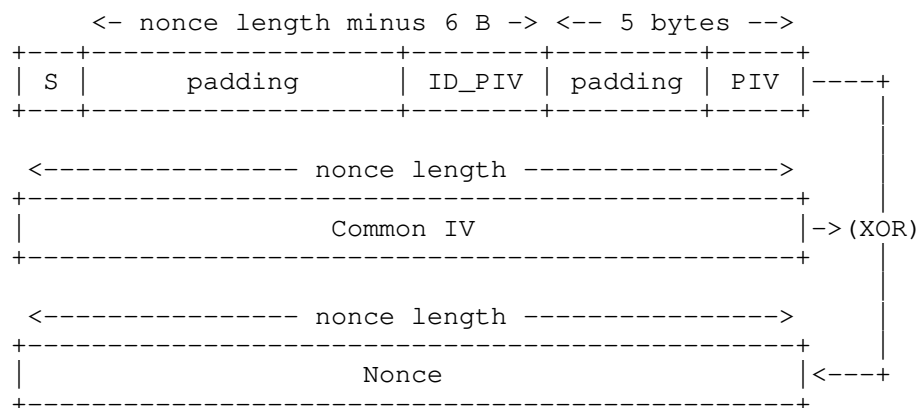


Figure 8: AEAD Nonce Formation

5.3. Plaintext

The plaintext is formatted as a CoAP message without Header (see Figure 9) consisting of:

- o the Code of the original CoAP message as defined in Section 3 of [RFC7252]; and
- o all Inner option message fields (see Section 4.1.1) present in the original CoAP message (see Section 4.1). The options are encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of Class E option; and
- o the Payload of original CoAP message, if present, and in that case prefixed by the one-byte Payload Marker (0xff).

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

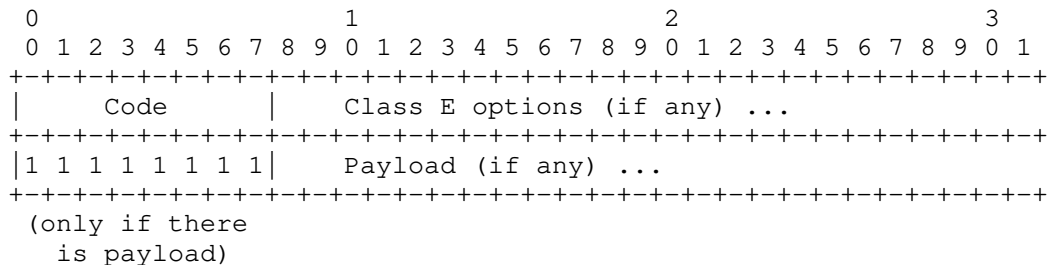


Figure 9: Plaintext

5.4. Additional Authenticated Data

The `external_aad` SHALL be a CBOR array wrapped in a `bstr` object as defined below:

```

external_aad = bstr .cbor aad_array

aad_array = [
  oscore_version : uint,
  algorithms : [ alg_aead : int / tstr ],
  request_kid : bstr,
  request_piv : bstr,
  options : bstr,
]

```

where:

- o `oscore_version`: contains the OSCORE version number. Implementations of this specification MUST set this field to 1. Other values are reserved for future versions.
- o `algorithms`: contains (for extensibility) an array of algorithms, according to this specification only containing `alg_aead`.
- o `alg_aead`: contains the AEAD Algorithm from the security context used for the exchange (see Section 3.1).
- o `request_kid`: contains the value of the 'kid' in the COSE object of the request (see Section 5).
- o `request_piv`: contains the value of the 'Partial IV' in the COSE object of the request (see Section 5).

- o options: contains the Class I options (see Section 4.1.2) present in the original CoAP message encoded as described in Section 3.1 of [RFC7252], where the delta is the difference to the previously included instance of class I option.

The `oscore_version` and `algorithms` parameters are established out-of-band and are thus never transported in OSCORE, but the `external_aad` allows to verify that they are the same in both endpoints.

NOTE: The format of the `external_aad` is for simplicity the same for requests and responses, although some parameters, e.g. `request_kid`, need not be integrity protected in all requests.

The Additional Authenticated Data (AAD) is composed from the `external_aad` as described in Section 5.3 of [RFC8152]:

```
AAD = Enc_structure = [ "Encrypt0", h'', external_aad ]
```

The following is an example of AAD constructed using AEAD Algorithm = AES-CCM-16-64-128 (10), `request_kid` = 0x00, `request_piv` = 0x25 and no Class I options:

- o `oscore_version`: 0x01 (1 byte)
- o `algorithms`: 0x810a (2 bytes)
- o `request_kid`: 0x00 (1 byte)
- o `request_piv`: 0x25 (1 byte)
- o `options`: 0x (0 bytes)
- o `aad_array`: 0x8501810a4100412540 (9 bytes)
- o `external_aad`: 0x498501810a4100412540 (10 bytes)
- o `AAD`: 0x8368456e63727970743040498501810a4100412540 (21 bytes)

Note that the AAD consists of a fixed string of 11 bytes concatenated with the `external_aad`.

6. OSCORE Header Compression

The Concise Binary Object Representation (CBOR) [RFC7049] combines very small message sizes with extensibility. The CBOR Object Signing and Encryption (COSE) [RFC8152] uses CBOR to create compact encoding of signed and encrypted data. COSE is however constructed to support a large number of different stateless use cases, and is not fully

optimized for use as a stateful security protocol, leading to a larger than necessary message expansion. In this section, we define a stateless header compression mechanism, simply removing redundant information from the COSE objects, which significantly reduces the per-packet overhead. The result of applying this mechanism to a COSE object is called the "compressed COSE object".

The COSE_Encrypt0 object used in OSCORE is transported in the OSCORE option and in the Payload. The Payload contains the Ciphertext of the COSE object. The headers of the COSE object are compactly encoded as described in the next section.

6.1. Encoding of the OSCORE Option Value

The value of the OSCORE option SHALL contain the OSCORE flag bits, the Partial IV parameter, the 'kid context' parameter (length and value), and the 'kid' parameter as follows:

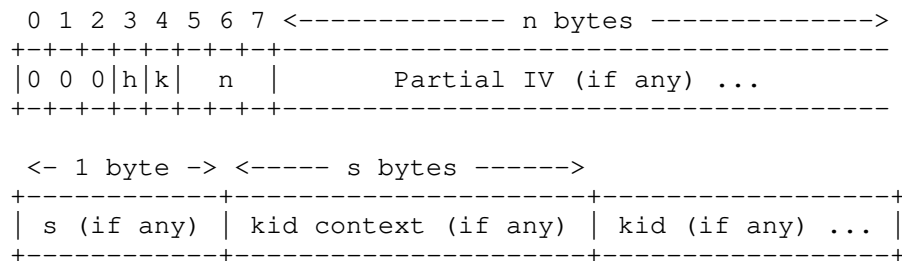


Figure 10: The OSCORE Option Value

- o The first byte, containing the OSCORE flag bits, encodes the following set of bits and the length of the Partial IV parameter:
 - * The three least significant bits encode the Partial IV length n. If n = 0 then the Partial IV is not present in the compressed COSE object. The values n = 6 and n = 7 are reserved.
 - * The fourth least significant bit is the 'kid' flag, k: it is set to 1 if the kid is present in the compressed COSE object.
 - * The fifth least significant bit is the 'kid context' flag, h: it is set to 1 if the compressed COSE object contains a 'kid context' (see Section 5.1).
 - * The sixth to eighth least significant bits are reserved for future use. These bits SHALL be set to zero when not in use. According to this specification, if any of these bits are set

to 1 the message is considered to be malformed and decompression fails as specified in item 2 of Section 8.2.

The flag bits are registered in the OSCORE Flag Bits registry specified in Section 13.7.

- o The following n bytes encode the value of the Partial IV, if the Partial IV is present ($n > 0$).
- o The following 1 byte encode the length of the 'kid context' (Section 5.1) s , if the 'kid context' flag is set ($h = 1$).
- o The following s bytes encode the 'kid context', if the 'kid context' flag is set ($h = 1$).
- o The remaining bytes encode the value of the 'kid', if the 'kid' is present ($k = 1$).

Note that the 'kid' MUST be the last field of the OSCORE option value, even in case reserved bits are used and additional fields are added to it.

The length of the OSCORE option thus depends on the presence and length of Partial IV, 'kid context', 'kid', as specified in this section, and on the presence and length of the other parameters, as defined in the separate documents.

6.2. Encoding of the OSCORE Payload

The payload of the OSCORE message SHALL encode the ciphertext of the COSE object.

6.3. Examples of Compressed COSE Objects

This section covers a list of OSCORE Header Compression examples for requests and responses. The examples assume the COSE_Encrypt0 object is set (which means the CoAP message and cryptographic material is known). Note that the full CoAP unprotected message, as well as the full security context, is not reported in the examples, but only the input necessary to the compression mechanism, i.e. the COSE_Encrypt0 object. The output is the compressed COSE object as defined in Section 6, divided into two parts, since the object is transported in two CoAP fields: OSCORE option and payload.

1. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = 0x25, and Partial IV = 0x05

Before compression (24 bytes):

```
[
  h'',
  { 4:h'25', 6:h'05' },
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (17 bytes):

Flag byte: 0b00001001 = 0x09 (1 byte)

Option Value: 0x090525 (3 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

2. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = empty string, and Partial IV = 0x00

Before compression (23 bytes):

```
[
  h'',
  { 4:h'', 6:h'00' },
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (16 bytes):

Flag byte: 0b00001001 = 0x09 (1 byte)

Option Value: 0x0900 (2 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

3. Request with ciphertext = 0xaea0155667924dff8a24e4cb35b9, kid = empty string, Partial IV = 0x05, and kid context = 0x44616c656b

Before compression (30 bytes):

```
[
  h'',
  { 4:h'', 6:h'05', 8:h'44616c656b' },
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (22 bytes):

Flag byte: 0b00011001 = 0x19 (1 byte)

Option Value: 0x19050544616c656b (8 bytes)

Payload: 0xae a0155667924dff8a24e4cb35b9 (14 bytes)

4. Response with ciphertext = 0xaea0155667924dff8a24e4cb35b9 and no Partial IV

Before compression (18 bytes):

```
[
  h'',
  {},
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (14 bytes):

Flag byte: 0b00000000 = 0x00 (1 byte)

Option Value: 0x (0 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

5. Response with ciphertext = 0xaea0155667924dff8a24e4cb35b9 and Partial IV = 0x07

Before compression (21 bytes):

```
[
  h'',
  { 6:h'07' },
  h'aea0155667924dff8a24e4cb35b9',
]
```

After compression (16 bytes):

Flag byte: 0b00000001 = 0x01 (1 byte)

Option Value: 0x0107 (2 bytes)

Payload: 0xaea0155667924dff8a24e4cb35b9 (14 bytes)

7. Message Binding, Sequence Numbers, Freshness, and Replay Protection

7.1. Message Binding

In order to prevent response delay and mismatch attacks [I-D.mattsson-core-coap-actuators] from on-path attackers and compromised intermediaries, OSCORE binds responses to the requests by including the 'kid' and Partial IV of the request in the AAD of the response. The server therefore needs to store the 'kid' and Partial IV of the request until all responses have been sent.

7.2. Sequence Numbers

An AEAD nonce MUST NOT be used more than once per AEAD key. The uniqueness of (key, nonce) pairs is shown in Appendix D.4, and in particular depends on a correct usage of Partial IVs (which encode the Sender Sequence Numbers, see Section 5). If messages are processed concurrently, the operation of reading and increasing the Sender Sequence Number MUST be atomic.

7.2.1. Maximum Sequence Number

The maximum Sender Sequence Number is algorithm dependent (see Section 12), and SHALL be less than 2^{40} . If the Sender Sequence Number exceeds the maximum, the endpoint MUST NOT process any more messages with the given Sender Context. If necessary, the endpoint SHOULD acquire a new security context before this happens. The latter is out of scope of this document.

7.3. Freshness

For requests, OSCORE provides only the guarantee that the request is not older than the security context. For applications having stronger demands on request freshness (e.g., control of actuators), OSCORE needs to be augmented with mechanisms providing freshness, for example as specified in [I-D.ietf-core-echo-request-tag].

Assuming an honest server (see Appendix D), the message binding guarantees that a response is not older than its request. For responses that are not notifications (i.e. when there is a single response to a request), this gives absolute freshness. For notifications, the absolute freshness gets weaker with time, and it is RECOMMENDED that the client regularly re-register the observation. Note that the message binding does not guarantee that misbehaving server created the response before receiving the request, i.e. it does not verify server aliveness.

For requests and notifications, OSCORE also provides relative freshness in the sense that the received Partial IV allows a recipient to determine the relative order of requests or responses.

7.4. Replay Protection

In order to protect from replay of requests, the server's Recipient Context includes a Replay Window. A server SHALL verify that a Partial IV = Sender Sequence Number received in the COSE object has not been received before. If this verification fails, the server SHALL stop processing the message, and MAY optionally respond with a 4.01 (Unauthorized) error message. Also, the server MAY set an Outer Max-Age option with value zero, to inform any intermediary that the response is not to be cached. The diagnostic payload MAY contain the "Replay detected" string. The size and type of the Replay Window depends on the use case and the protocol with which the OSCORE message is transported. In case of reliable and ordered transport from endpoint to endpoint, e.g. TCP, the server MAY just store the last received Partial IV and require that newly received Partial IVs equals the last received Partial IV + 1. However, in case of mixed reliable and unreliable transports and where messages may be lost, such a replay mechanism may be too restrictive and the default replay window be more suitable (see Section 3.2.2).

Responses (with or without Partial IV) are protected against replay as they are bound to the request and the fact that only a single response is accepted. Note that the Partial IV is not used for replay protection in this case.

The operation of validating the Partial IV and updating the replay protection MUST be atomic.

7.4.1. Replay Protection of Notifications

The following applies additionally when Observe is supported.

The Notification Number is initialized to the Partial IV of the first successfully verified notification in response to the registration request. A client MUST only accept at most one Observe notifications without Partial IV, and treat it as the oldest notification received. A client receiving a notification containing a Partial IV SHALL compare the Partial IV with the Notification Number associated to that Observe registration. The client MUST stop processing notifications with a Partial IV which has been previously received. Applications MAY decide that a client only processes notifications which have greater Partial IV than the Notification Number.

If the verification of the response succeeds, and the received Partial IV was greater than the Notification Number then the client SHALL overwrite the corresponding Notification Number with the received Partial IV.

7.5. Losing Part of the Context State

To prevent reuse of an AEAD nonce with the same AEAD key, or from accepting replayed messages, an endpoint needs to handle the situation of losing rapidly changing parts of the context, such as the Sender Sequence Number, and Replay Window. These are typically stored in RAM and therefore lost in the case of e.g. an unplanned reboot. There are different alternatives to recover, for example:

1. The endpoints can reuse an existing Security Context after updating the mutable parts of the security context (Sender Sequence Number, and Replay Window). This requires that the mutable parts of the security context are available throughout the lifetime of the device, or that the device can establish safe security context after loss of mutable security context data. Examples is given based on careful use of non-volatile memory, see Appendix B.1.1, and additionally the use of the Echo option, see Appendix B.1.2. If an endpoint makes use of a partial security context stored in non-volatile memory, it MUST NOT reuse a previous Sender Sequence Number and MUST NOT accept previously received messages.
2. The endpoints can reuse an existing shared Master Secret and derive new Sender and Recipient Contexts, see Appendix B.2 for an example. This typically requires a good source of randomness.
3. The endpoints can use a trusted-third party assisted key establishment protocol such as [I-D.ietf-ace-oscore-profile]. This requires the execution of three-party protocol and may require a good source of randomness.
4. The endpoints can run a key exchange protocol providing forward secrecy resulting in a fresh Master Secret, from which an entirely new Security Context is derived. This requires a good source of randomness, and additionally, the transmission and processing of the protocol may have a non-negligible cost, e.g. in terms of power consumption.

The endpoints need to be configured with information about which method is used. The choice of method may depend on capabilities of the devices deployed and the solution architecture. Using a key exchange protocol is necessary for deployments that require forward secrecy.

8. Processing

This section describes the OSCORE message processing. Additional processing for Observe or Block-wise are described in subsections.

Note that, analogously to [RFC7252] where the Token and source/destination pair are used to match a response with a request, both endpoints MUST keep the association (Token, {Security Context, Partial IV of the request}), in order to be able to find the Security Context and compute the AAD to protect or verify the response. The association MAY be forgotten after it has been used to successfully protect or verify the response, with the exception of Observe processing, where the association MUST be kept as long as the Observation is active.

The processing of the Sender Sequence Number follows the procedure described in Section 3 of [I-D.mcgregor-iv-gen].

8.1. Protecting the Request

Given a CoAP request, the client SHALL perform the following steps to create an OSCORE request:

1. Retrieve the Sender Context associated with the target resource.
2. Compose the Additional Authenticated Data and the plaintext, as described in Sections 5.3 and 5.4.
3. Encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV as described in Section 5.2.
4. Encrypt the COSE object using the Sender Key. Compress the COSE Object as specified in Section 6.
5. Format the OSCORE message according to Section 4. The OSCORE option is added (see Section 4.1.2).

8.2. Verifying the Request

A server receiving a request containing the OSCORE option SHALL perform the following steps:

1. Discard Code and all class E options (marked in Figure 5 with 'x' in column E) present in the received message. For example, an If-Match Outer option is discarded, but an Uri-Host Outer option is not discarded.

2. Decompress the COSE Object (Section 6) and retrieve the Recipient Context associated with the Recipient ID in the 'kid' parameter, additionally using the 'kid context', if present. If either the decompression or the COSE message fails to decode, or the server fails to retrieve a Recipient Context with Recipient ID corresponding to the 'kid' parameter received, then the server SHALL stop processing the request.
 - * If either the decompression or the COSE message fails to decode, the server MAY respond with a 4.02 (Bad Option) error message. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain the string "Failed to decode COSE".
 - * If the server fails to retrieve a Recipient Context with Recipient ID corresponding to the 'kid' parameter received, the server MAY respond with a 4.01 (Unauthorized) error message. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain the string "Security context not found".
3. Verify that the 'Partial IV' has not been received before using the Replay Window, as described in Section 7.4.
4. Compose the Additional Authenticated Data, as described in Section 5.4.
5. Compute the AEAD nonce from the Recipient ID, Common IV, and the 'Partial IV' parameter, received in the COSE Object.
6. Decrypt the COSE object using the Recipient Key, as per [RFC8152] Section 5.3. (The decrypt operation includes the verification of the integrity.)
 - * If decryption fails, the server MUST stop processing the request and MAY respond with a 4.00 (Bad Request) error message. The server MAY set an Outer Max-Age option with value zero. The diagnostic payload MAY contain the "Decryption failed" string.
 - * If decryption succeeds, update the Replay Window, as described in Section 7.
7. Add decrypted Code, options, and payload to the decrypted request. The OSCORE option is removed.
8. The decrypted CoAP request is processed according to [RFC7252].

8.2.1. Supporting Block-wise

If Block-wise is supported, insert the following step before any other:

A. If Block-wise is present in the request, then process the Outer Block options according to [RFC7959], until all blocks of the request have been received (see Section 4.1.3.4).

8.3. Protecting the Response

If a CoAP response is generated in response to an OSCORE request, the server SHALL perform the following steps to create an OSCORE response. Note that CoAP error responses derived from CoAP processing (step 8 in Section 8.2) are protected, as well as successful CoAP responses, while the OSCORE errors (steps 2, 3, and 6 in Section 8.2) do not follow the processing below, but are sent as simple CoAP responses, without OSCORE processing.

1. Retrieve the Sender Context in the Security Context associated with the Token.
2. Compose the Additional Authenticated Data and the plaintext, as described in Sections 5.3 and 5.4.
3. Compute the AEAD nonce as described in Section 5.2:
 - * Either use the AEAD nonce from the request, or
 - * Encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV.
4. Encrypt the COSE object using the Sender Key. Compress the COSE Object as specified in Section 6. If the AEAD nonce was constructed from a new Partial IV, this Partial IV MUST be included in the message. If the AEAD nonce from the request was used, the Partial IV MUST NOT be included in the message.
5. Format the OSCORE message according to Section 4. The OSCORE option is added (see Section 4.1.2).

8.3.1. Supporting Observe

If Observe is supported, insert the following step between step 2 and 3 of Section 8.3:

- A. If the response is an observe notification:
 - o If the response is the first notification:
 - * compute the AEAD nonce as described in Section 5.2:
 - + Either use the AEAD nonce from the request, or
 - + Encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV.
 - Then go to 4.
- o If the response is not the first notification:
 - * encode the Partial IV (Sender Sequence Number in network byte order) and increment the Sender Sequence Number by one. Compute the AEAD nonce from the Sender ID, Common IV, and Partial IV, then go to 4.

8.4. Verifying the Response

A client receiving a response containing the OSCORE option SHALL perform the following steps:

1. Discard Code and all class E options (marked in Figure 5 with 'x' in column E) present in the received message. For example, ETag Outer option is discarded, as well as Max-Age Outer option.
2. Retrieve the Recipient Context in the Security Context associated with the Token. Decompress the COSE Object (Section 6). If either the decompression or the COSE message fails to decode, then go to 8.
3. Compose the Additional Authenticated Data, as described in Section 5.4.
4. Compute the AEAD nonce
 - * If the Partial IV is not present in the response, the AEAD nonce from the request is used.
 - * If the Partial IV is present in the response, compute the AEAD nonce from the Recipient ID, Common IV, and the 'Partial IV' parameter, received in the COSE Object.

5. Decrypt the COSE object using the Recipient Key, as per [RFC8152] Section 5.3. (The decrypt operation includes the verification of the integrity.) If decryption fails, then go to 8.
6. Add decrypted Code, options and payload to the decrypted request. The OSCORE option is removed.
7. The decrypted CoAP response is processed according to [RFC7252].
8. In case any of the previous erroneous conditions apply: the client SHALL stop processing the response.

8.4.1. Supporting Block-wise

If Block-wise is supported, insert the following step before any other:

- A. If Block-wise is present in the request, then process the Outer Block options according to [RFC7959], until all blocks of the request have been received (see Section 4.1.3.4).

8.4.2. Supporting Observe

If Observe is supported:

Insert the following step between step 5 and step 6:

- A. If the request was an Observe registration, then:
 - o If the Partial IV is not present in the response, and Inner Observe is present, and the AEAD nonce from the request was already used once, then go to 8.
 - o If the Partial IV is present in the response and Inner Observe is present, then follow the processing described in Section 4.1.3.5.2 and Section 7.4.1, then:
 - * initialize the Notification Number (if first successfully verified notification), or
 - * overwrite the Notification Number (if the received Partial IV was greater than the Notification Number).

Replace step 8 of Section 8.4 with:

- B. In case any of the previous erroneous conditions apply: the client SHALL stop processing the response. An error condition occurring while processing a response to an observation request does

not cancel the observation. A client **MUST NOT** react to failure by re-registering the observation immediately.

9. Web Linking

The use of OSCORE **MAY** be indicated by a target attribute "osc" in a web link [RFC8288] to a resource, e.g. using a link-format document [RFC6690] if the resource is accessible over CoAP.

The "osc" attribute is a hint indicating that the destination of that link is only accessible using OSCORE, and unprotected access to it is not supported. Note that this is simply a hint, it does not include any security context material or any other information required to run OSCORE.

A value **MUST NOT** be given for the "osc" attribute; any present value **MUST** be ignored by parsers. The "osc" attribute **MUST NOT** appear more than once in a given link-value; occurrences after the first **MUST** be ignored by parsers.

The example in Figure 11 shows a use of the "osc" attribute: the client does resource discovery on a server, and gets back a list of resources, one of which includes the "osc" attribute indicating that the resource is protected with OSCORE. The link-format notation (see Section 5 of [RFC6690]) is used.

```
REQ: GET /.well-known/core
RES: 2.05 Content
    </sensors/temp>;osc,
    </sensors/light>;if="sensor"
```

Figure 11: The web link

10. CoAP-to-CoAP Forwarding Proxy

CoAP is designed for proxy operations (see Section 5.7 of [RFC7252]).

OSCORE is designed to work with OSCORE-unaware CoAP proxies. Security requirements for forwarding are listed in Section 2.2.1 of [I-D.hartke-core-e2e-security-reqs]. Proxy processing of the (Outer) Proxy-Uri option works as defined in [RFC7252]. Proxy processing of the (Outer) Block options works as defined in [RFC7959].

However, not all CoAP proxy operations are useful:

- o Since a CoAP response is only applicable to the original CoAP request, caching is in general not useful. In support of existing

proxies, OSCORE uses the outer Max-Age option, see Section 4.1.3.1.

- o Proxy processing of the (Outer) Observe option as defined in [RFC7641] is specified in Section 4.1.3.5.

Optionally, a CoAP proxy MAY detect OSCORE and act accordingly. An OSCORE-aware CoAP proxy:

- o SHALL bypass caching for the request if the OSCORE option is present
- o SHOULD avoid caching responses to requests with an OSCORE option

In the case of Observe (see Section 4.1.3.5) the OSCORE-aware CoAP proxy:

- o SHALL NOT initiate an Observe registration
- o MAY verify the order of notifications using Partial IV rather than the Observe option

11. HTTP Operations

The CoAP request/response model may be mapped to HTTP and vice versa as described in Section 10 of [RFC7252]. The HTTP-CoAP mapping is further detailed in [RFC8075]. This section defines the components needed to map and transport OSCORE messages over HTTP hops. By mapping between HTTP and CoAP and by using cross-protocol proxies OSCORE may be used end-to-end between e.g. an HTTP client and a CoAP server. Examples are provided at the end of the section.

11.1. The HTTP OSCORE Header Field

The HTTP OSCORE Header Field (see Section 13.4) is used for carrying the content of the CoAP OSCORE option when transporting OSCORE messages over HTTP hops.

The HTTP OSCORE header field is only used in POST requests and 200 (OK) responses. When used, the HTTP header field Content-Type is set to 'application/oscore' (see Section 13.5) indicating that the HTTP body of this message contains the OSCORE payload (see Section 6.2). No additional semantics is provided by other message fields.

Using the Augmented Backus-Naur Form (ABNF) notation of [RFC5234], including the following core ABNF syntax rules defined by that specification: ALPHA (letters) and DIGIT (decimal digits), the HTTP OSCORE header field value is as follows.

base64url-char = ALPHA / DIGIT / "-" / "_"

OSCORE = 2*base64url-char

The HTTP OSCORE header field is not appropriate to list in the Connection header field (see Section 6.1 of [RFC7230]) since it is not hop-by-hop. OSCORE messages are generally not useful when served from cache (i.e., they will generally be marked Cache-Control: no-cache) and so interaction with Vary is not relevant (Section 7.1.4 of [RFC7231]). Since the HTTP OSCORE header field is critical for message processing, moving it from headers to trailers renders the message unusable in case trailers are ignored (see Section 4.1 of [RFC7230]).

Intermediaries are in general not allowed to insert, delete, or modify the OSCORE header. Changes to the HTTP OSCORE header field will in general violate the integrity of the OSCORE message resulting in an error. For the same reason the HTTP OSCORE header field is in general not preserved across redirects.

Since redirects are not defined in the mappings between HTTP and CoAP [RFC8075][RFC7252], a number of conditions need to be fulfilled for redirects to work. For CoAP client to HTTP server, such conditions include:

- o the CoAP-to-HTTP proxy follows the redirect, instead of the CoAP client as in the HTTP case
- o the CoAP-to-HTTP proxy copies the HTTP OSCORE header field and body to the new request
- o the target of the redirect has the necessary OSCORE security context required to decrypt and verify the message

Since OSCORE requires HTTP body to be preserved across redirects, the HTTP server is RECOMMENDED to reply with 307 or 308 instead of 301 or 302.

For the case of HTTP client to CoAP server, although redirect is not defined for CoAP servers [RFC7252], an HTTP client receiving a redirect should generate a new OSCORE request for the server it was redirected to.

11.2. CoAP-to-HTTP Mapping

Section 10.1 of [RFC7252] describes the fundamentals of the CoAP-to-HTTP cross-protocol mapping process. The additional rules for OSCORE messages are:

- o The HTTP OSCORE header field value is set to
 - * AA if the CoAP OSCORE option is empty, otherwise
 - * the value of the CoAP OSCORE option (Section 6.1) in base64url (Section 5 of [RFC4648]) encoding without padding. Implementation notes for this encoding are given in Appendix C of [RFC7515].
- o The HTTP Content-Type is set to 'application/oscore' (see Section 13.5), independent of CoAP Content-Format.

11.3. HTTP-to-CoAP Mapping

Section 10.2 of [RFC7252] and [RFC8075] specify the behavior of an HTTP-to-CoAP proxy. The additional rules for HTTP messages with the OSCORE header field are:

- o The CoAP OSCORE option is set as follows:
 - * empty if the value of the HTTP OSCORE header field is a single zero byte (0x00) represented by AA, otherwise
 - * the value of the HTTP OSCORE header field decoded from base64url (Section 5 of [RFC4648]) without padding. Implementation notes for this encoding are given in Appendix C of [RFC7515].
- o The CoAP Content-Format option is omitted, the content format for OSCORE (Section 13.6) MUST NOT be used.

11.4. HTTP Endpoints

Restricted to subsets of HTTP and CoAP supporting a bijective mapping, OSCORE can be originated or terminated in HTTP endpoints.

The sending HTTP endpoint uses [RFC8075] to translate the HTTP message into a CoAP message. The CoAP message is then processed with OSCORE as defined in this document. The OSCORE message is then mapped to HTTP as described in Section 11.2 and sent in compliance with the rules in Section 11.1.

The receiving HTTP endpoint maps the HTTP message to a CoAP message using [RFC8075] and Section 11.3. The resulting OSCORE message is processed as defined in this document. If successful, the plaintext CoAP message is translated to HTTP for normal processing in the endpoint.

11.5. Example: HTTP Client and CoAP Server

This section is giving an example of how a request and a response between an HTTP client and a CoAP server could look like. The example is not a test vector but intended as an illustration of how the message fields are translated in the different steps.

Mapping and notation here is based on "Simple Form" (Section 5.4.1 of [RFC8075]).

[HTTP request -- Before client object security processing]

```
GET http://proxy.url/hc/?target_uri=coap://server.url/orders
HTTP/1.1
```

[HTTP request -- HTTP Client to Proxy]

```
POST http://proxy.url/hc/?target_uri=coap://server.url/ HTTP/1.1
Content-Type: application/oscore
OSCORE: CSU
Body: 09 07 01 13 61 f7 0f d2 97 b1 [binary]
```

[CoAP request -- Proxy to CoAP Server]

```
POST coap://server.url/
OSCORE: 09 25
Payload: 09 07 01 13 61 f7 0f d2 97 b1 [binary]
```

[CoAP request -- After server object security processing]

```
GET coap://server.url/orders
```

[CoAP response -- Before server object security processing]

```
2.05 Content
Content-Format: 0
Payload: Exterminate! Exterminate!
```

[CoAP response -- CoAP Server to Proxy]

```
2.04 Changed
OSCORE: [empty]
Payload: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]
```

[HTTP response -- Proxy to HTTP Client]

```
HTTP/1.1 200 OK
Content-Type: application/oscore
OSCORE: AA
Body: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]
```

[HTTP response -- After client object security processing]

```
HTTP/1.1 200 OK
Content-Type: text/plain
Body: Exterminate! Exterminate!
```

Note that the HTTP Status Code 200 in the next-to-last message is the mapping of CoAP Code 2.04 (Changed), whereas the HTTP Status Code 200 in the last message is the mapping of the CoAP Code 2.05 (Content), which was encrypted within the compressed COSE object carried in the Body of the HTTP response.

11.6. Example: CoAP Client and HTTP Server

This section is giving an example of how a request and a response between a CoAP client and an HTTP server could look like. The example is not a test vector but intended as an illustration of how the message fields are translated in the different steps

[CoAP request -- Before client object security processing]

```
GET coap://proxy.url/
Proxy-Uri=http://server.url/orders
```

[CoAP request -- CoAP Client to Proxy]

```
POST coap://proxy.url/
Proxy-Uri=http://server.url/
OSCORE: 09 25
Payload: 09 07 01 13 61 f7 0f d2 97 b1 [binary]
```

[HTTP request -- Proxy to HTTP Server]

```
POST http://server.url/ HTTP/1.1
Content-Type: application/oscore
OSCORE: CSU
Body: 09 07 01 13 61 f7 0f d2 97 b1 [binary]
```

[HTTP request -- After server object security processing]

```
GET http://server.url/orders HTTP/1.1
```


[HTTP response -- Before server object security processing]

HTTP/1.1 200 OK
Content-Type: text/plain
Body: Exterminate! Exterminate!

[HTTP response -- HTTP Server to Proxy]

HTTP/1.1 200 OK
Content-Type: application/oscore
OSCORE: AA
Body: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[CoAP response -- Proxy to CoAP Client]

2.04 Changed
OSCORE: [empty]
Payload: 00 31 d1 fc f6 70 fb 0c 1d d5 ... [binary]

[CoAP response -- After client object security processing]

2.05 Content
Content-Format: 0
Payload: Exterminate! Exterminate!

Note that the HTTP Code 2.04 (Changed) in the next-to-last message is the mapping of HTTP Status Code 200, whereas the CoAP Code 2.05 (Content) in the last message is the value that was encrypted within the compressed COSE object carried in the Body of the HTTP response.

12. Security Considerations

An overview of the security properties is given in Appendix D.

12.1. End-to-end Protection

In scenarios with intermediary nodes such as proxies or gateways, transport layer security such as (D)TLS only protects data hop-by-hop. As a consequence, the intermediary nodes can read and modify any information. The trust model where all intermediary nodes are considered trustworthy is problematic, not only from a privacy perspective, but also from a security perspective, as the intermediaries are free to delete resources on sensors and falsify commands to actuators (such as "unlock door", "start fire alarm", "raise bridge"). Even in the rare cases where all the owners of the intermediary nodes are fully trusted, attacks and data breaches make such an architecture brittle.

(D)TLS protects hop-by-hop the entire message. OSCORE protects end-to-end all information that is not required for proxy operations (see Section 4). (D)TLS and OSCORE can be combined, thereby enabling end-to-end security of the message payload, in combination with hop-by-hop protection of the entire message, during transport between endpoint and intermediary node. In particular when OSCORE is used with HTTP, the additional TLS protection of HTTP hops is RECOMMENDED, e.g. between an HTTP endpoint and a proxy translating between HTTP and CoAP.

Applications need to consider that certain message fields and messages types are not protected end-to-end and may be spoofed or manipulated. The consequences of unprotected message fields are analyzed in Appendix D.5.

12.2. Security Context Establishment

The use of COSE_Encrypt0 and AEAD to protect messages as specified in this document requires an established security context. The method to establish the security context described in Section 3.2 is based on a common Master Secret and unique Sender IDs. The necessary input parameters may be pre-established or obtained using a key establishment protocol augmented with establishment of Sender/Recipient ID, such as a key exchange protocol or the OSCORE profile of the ACE framework [I-D.ietf-ace-oscore-profile]. Such a procedure must ensure that the requirements of the security context parameters for the intended use are complied with (see Section 3.3) and also in error situations. While recipient IDs are allowed to coincide between different security contexts (see Section 3.3), this may cause a server to process multiple verifications before finding the right security context or rejecting a message. Considerations for deploying OSCORE with a fixed Master Secret are given in Appendix B.

12.3. Master Secret

OSCORE uses HKDF [RFC5869] and the established input parameters to derive the security context. The required properties of the security context parameters are discussed in Section 3.3, in this section we focus on the Master Secret. HKDF denotes in this specification the composition of the expand and extract functions as defined in [RFC5869] and the Master Secret is used as Input Key Material (IKM).

Informally, HKDF takes as source an IKM containing some good amount of randomness but not necessarily distributed uniformly (or for which an attacker has some partial knowledge) and derive from it one or more cryptographically strong secret keys [RFC5869].

Therefore, the main requirement for the OSCORE Master Secret, in addition to being secret, is that it has a good amount of randomness. The selected key establishment schemes must ensure that the necessary properties for the Master Secret are fulfilled. For pre-shared key deployments and key transport solutions such as [I-D.ietf-ace-oscore-profile], the Master Secret can be generated offline using a good random number generator. Randomness requirements for security are described in [RFC4086].

12.4. Replay Protection

Replay attacks need to be considered in different parts of the implementation. Most AEAD algorithms require a unique nonce for each message, for which the sender sequence numbers in the COSE message field 'Partial IV' is used. If the recipient accepts any sequence number larger than the one previously received, then the problem of sequence number synchronization is avoided. With reliable transport, it may be defined that only messages with sequence number which are equal to previous sequence number + 1 are accepted. An adversary may try to induce a device reboot for the purpose of replaying a message (see Section 7.5).

Note that sharing a security context between servers may open up for replay attacks, for example if the replay windows are not synchronized.

12.5. Client Aliveness

A verified OSCORE request enables the server to verify the identity of the entity who generated the message. However, it does not verify that the client is currently involved in the communication, since the message may be a delayed delivery of a previously generated request which now reaches the server. To verify the aliveness of the client the server may use the Echo option in the response to a request from the client (see [I-D.ietf-core-echo-request-tag]).

12.6. Cryptographic Considerations

The maximum sender sequence number is dependent on the AEAD algorithm. The maximum sender sequence number is $2^{40} - 1$, or any algorithm specific lower limit, after which a new security context must be generated. The mechanism to build the AEAD nonce (Section 5.2) assumes that the nonce is at least 56 bits, and the Partial IV is at most 40 bits. The mandatory-to-implement AEAD algorithm AES-CCM-16-64-128 is selected for compatibility with CCM*. AEAD algorithms that require unpredictable nonces are not supported.

In order to prevent cryptanalysis when the same plaintext is repeatedly encrypted by many different users with distinct AEAD keys, the AEAD nonce is formed by mixing the sequence number with a secret per-context initialization vector (Common IV) derived along with the keys (see Section 3.1 of [RFC8152]), and by using a Master Salt in the key derivation (see [MF00] for an overview). The Master Secret, Sender Key, Recipient Key, and Common IV must be secret, the rest of the parameters may be public. The Master Secret must have a good amount of randomness (see Section 12.3).

The ID Context, Sender ID, and Partial IV are always at least implicitly integrity protected, as manipulation leads to the wrong nonce or key being used and therefore results in decryption failure.

12.7. Message Segmentation

The Inner Block options enable the sender to split large messages into OSCORE-protected blocks such that the receiving endpoint can verify blocks before having received the complete message. The Outer Block options allow for arbitrary proxy fragmentation operations that cannot be verified by the endpoints, but can by policy be restricted in size since the Inner Block options allow for secure fragmentation of very large messages. A maximum message size (above which the sending endpoint fragments the message and the receiving endpoint discards the message, if complying to the policy) may be obtained as part of normal resource discovery.

12.8. Privacy Considerations

Privacy threats executed through intermediary nodes are considerably reduced by means of OSCORE. End-to-end integrity protection and encryption of the message payload and all options that are not used for proxy operations, provide mitigation against attacks on sensor and actuator communication, which may have a direct impact on the personal sphere.

The unprotected options (Figure 5) may reveal privacy sensitive information, see Appendix D.5. CoAP headers sent in plaintext allow, for example, matching of CON and ACK (CoAP Message Identifier), matching of request and responses (Token) and traffic analysis. OSCORE does not provide protection for HTTP header fields which are not both CoAP-mappable and class E. The HTTP message fields which are visible to on-path entity are only used for the purpose of transporting the OSCORE message, whereas the application layer message is encoded in CoAP and encrypted.

COSE message fields, i.e. the OSCORE option, may reveal information about the communicating endpoints. E.g. 'kid' and 'kid context',

which are intended to help the server find the right context, may reveal information about the client. Tracking 'kid' and 'kid context' to one server may be used for correlating requests from one client.

Unprotected error messages reveal information about the security state in the communication between the endpoints. Unprotected signaling messages reveal information about the reliable transport used on a leg of the path. Using the mechanisms described in Section 7.5 may reveal when a device goes through a reboot. This can be mitigated by the device storing the precise state of sender sequence number and replay window on a clean shutdown.

The length of message fields can reveal information about the message. Applications may use a padding scheme to protect against traffic analysis.

13. IANA Considerations

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

Note to IANA: Please note all occurrences of "TBD1" in this specification should be assigned the same number.

13.1. COSE Header Parameters Registry

The 'kid context' parameter is added to the "COSE Header Parameters Registry":

- o Name: kid context
- o Label: TBD2
- o Value Type: bstr
- o Value Registry:
- o Description: Identifies the context for 'kid'
- o Reference: Section 5.1 of this document

Note to IANA: Label assignment in (Integer value between 1 and 255) is requested. (RFC Editor: Delete this note after IANA assignment)

13.2. CoAP Option Numbers Registry

The OSCORE option is added to the CoAP Option Numbers registry:

Number	Name	Reference
TBD1	OSCORE	[[this document]]

Note to IANA: Label assignment in (Integer value between 0 and 12) is requested. We also request Expert review if possible, to make sure a correct number for the option is selected (RFC Editor: Delete this note after IANA assignment)

Furthermore, the following existing entries in the CoAP Option Numbers registry are updated with a reference to the document specifying OSCORE processing of that option:

Number	Name	Reference
1	If-Match	[RFC7252] [[this document]]
3	Uri-Host	[RFC7252] [[this document]]
4	ETag	[RFC7252] [[this document]]
5	If-None-Match	[RFC7252] [[this document]]
6	Observe	[RFC7641] [[this document]]
7	Uri-Port	[RFC7252] [[this document]]
8	Location-Path	[RFC7252] [[this document]]
11	Uri-Path	[RFC7252] [[this document]]
12	Content-Format	[RFC7252] [[this document]]
14	Max-Age	[RFC7252] [[this document]]
15	Uri-Query	[RFC7252] [[this document]]
17	Accept	[RFC7252] [[this document]]
20	Location-Query	[RFC7252] [[this document]]
23	Block2	[RFC7959] [RFC8323] [[this document]]
27	Block1	[RFC7959] [RFC8323] [[this document]]
28	Size2	[RFC7959] [[this document]]
35	Proxy-Uri	[RFC7252] [[this document]]
39	Proxy-Scheme	[RFC7252] [[this document]]
60	Size1	[RFC7252] [[this document]]
258	No-Response	[RFC7967] [[this document]]

Future additions to the CoAP Option Numbers registry need to provide a reference to the document where the OSCORE processing of that CoAP Option is defined.

13.3. CoAP Signaling Option Numbers Registry

The OSCORE option is added to the CoAP Signaling Option Numbers registry:

Applies to	Number	Name	Reference
7.xx (all)	TBD1	OSCORE	[[this document]]

Note to IANA: The value in the "Number" field is the same value that's being assigned to the new Option Number. Please make sure TBD1 is not the same as any value in Numbers for any existing entry in the CoAP Signaling Option Numbers registry (at the time of writing this, that means make sure TBD1 is not 2 or 4) (RFC Editor: Delete this note after IANA assignment)

13.4. Header Field Registrations

The HTTP OSCORE header field is added to the Message Headers registry:

Header Field Name	Protocol	Status	Reference
OSCORE	http	standard	[[this document]], Section 11.1

13.5. Media Type Registrations

This section registers the 'application/oscore' media type in the "Media Types" registry. These media types are used to indicate that the content is an OSCORE message. The OSCORE body cannot be understood without the OSCORE header field value and the security context.

Type name: application

Subtype name: oscore

Required parameters: N/A

Optional parameters: N/A

Encoding considerations: binary

Security considerations: See the Security Considerations section of [[This document]].

Interoperability considerations: N/A

Published specification: [[This document]]

Applications that use this media type: IoT applications sending security content over HTTP(S) transports.

Fragment identifier considerations: N/A

Additional information:

- * Deprecated alias names for this type: N/A

- * Magic number(s): N/A

- * File extension(s): N/A

- * Macintosh file type code(s): N/A

Person & email address to contact for further information:
iesg@ietf.org

Intended usage: COMMON

Restrictions on usage: N/A

Author: Goeran Selander, goran.selander@ericsson.com

Change Controller: IESG

Provisional registration? No

13.6. CoAP Content-Formats Registry

Note to IANA: ID assignment in the 10000-64999 range is requested.
(RFC Editor: Delete this note after IANA assignment)

This section registers the media type 'application/oscore' media type in the "CoAP Content-Formats" registry. This Content-Format for the OSCORE payload is defined for potential future use cases and SHALL NOT be used in the OSCORE message. The OSCORE payload cannot be understood without the OSCORE option value and the security context.

Media Type	Encoding	ID	Reference
application/oscore		TBD3	[[this document]]

13.7. OSCORE Flag Bits Registry

This document defines a sub-registry for the OSCORE flag bits within the "CoRE Parameters" registry. The name of the sub-registry is "OSCORE Flag Bits". The registry should be created with the Expert Review policy. Guidelines for the experts are provided in Section 13.8.

The columns of the registry are:

- o bit position: This indicates the position of the bit in the set of OSCORE flag bits, starting at 0 for the most significant bit. The bit position must be an integer or a range of integers, in the range 0 to 63.
- o name: The name is present to make it easier to refer to and discuss the registration entry. The value is not used in the protocol. Names are to be unique in the table.
- o description: This contains a brief description of the use of the bit.
- o specification: This contains a pointer to the specification defining the entry.

The initial contents of the registry can be found in the table below. The specification column for all rows in that table should be this document. The entries with Bit Position of 0 and 1 are to be marked as 'Reserved'. The entry with Bit Position of 1 is going to be specified in a future document, and will be used to expand the space

for the OSCORE flag bits in Section 6.1, so that entries 8-63 of the registry are defined.

Bit Position	Name	Description	Specification
0	Reserved		
1	Reserved		
2	Unassigned		
3	Kid Context Flag	Set to 1 if 'kid context' is present in the compressed COSE object	[[this document]]
4	Kid Flag	Set to 1 if kid is present in the compressed COSE object	[[this document]]
5-7	Partial IV Length	Encodes the Partial IV length; can have value 0 to 5	[[this document]]
8-63	Unassigned		

13.8. Expert Review Instructions

The expert reviewers for the registry defined in this document are expected to ensure that the usage solves a valid use case that could not be solved better in a different way, that it is not going to duplicate one that is already registered, and that the registered point is likely to be used in deployments. They are furthermore expected to check the clarity of purpose and use of the requested code points. Experts should take into account the expected usage of entries when approving point assignment, and the length of the encoded value should be weighed against the number of code points left that encode to that size and the size of device it will be used on. Experts should block registration for entries 8-63 until these points are defined (i.e. until the mechanism for the OSCORE flag bits expansion via bit 1 is specified).

14. References

14.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <<https://www.rfc-editor.org/info/rfc4086>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<https://www.rfc-editor.org/info/rfc7049>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <<https://www.rfc-editor.org/info/rfc7231>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.

- [RFC7641] Hartke, K., "Observing Resources in the Constrained Application Protocol (CoAP)", RFC 7641, DOI 10.17487/RFC7641, September 2015, <<https://www.rfc-editor.org/info/rfc7641>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<https://www.rfc-editor.org/info/rfc7959>>.
- [RFC8075] Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for Mapping Implementations: HTTP to the Constrained Application Protocol (CoAP)", RFC 8075, DOI 10.17487/RFC8075, February 2017, <<https://www.rfc-editor.org/info/rfc8075>>.
- [RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <<https://www.rfc-editor.org/info/rfc8132>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.
- [RFC8323] Bormann, C., Lemay, S., Tschafenig, H., Hartke, K., Silverajan, B., and B. Raymor, Ed., "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC 8323, DOI 10.17487/RFC8323, February 2018, <<https://www.rfc-editor.org/info/rfc8323>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.

14.2. Informative References

- [I-D.bormann-6lo-coap-802-15-ie]
Bormann, C., "Constrained Application Protocol (CoAP) over IEEE 802.15.4 Information Element for IETF", draft-bormann-6lo-coap-802-15-ie-00 (work in progress), April 2016.
- [I-D.hartke-core-e2e-security-reqs]
Selander, G., Palombini, F., and K. Hartke, "Requirements for CoAP End-To-End Security", draft-hartke-core-e2e-security-reqs-03 (work in progress), July 2017.
- [I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-22 (work in progress), March 2019.
- [I-D.ietf-ace-oscore-profile]
Palombini, F., Seitz, L., Selander, G., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-ietf-ace-oscore-profile-07 (work in progress), February 2019.
- [I-D.ietf-cbor-cddl]
Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR and JSON data structures", draft-ietf-cbor-cddl-07 (work in progress), February 2019.
- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-03 (work in progress), October 2018.
- [I-D.ietf-core-oscore-groupcomm]
Tiloca, M., Selander, G., Palombini, F., and J. Park, "Group OSCORE - Secure Group Communication for CoAP", draft-ietf-core-oscore-groupcomm-03 (work in progress), October 2018.
- [I-D.mattsson-core-coap-actuators]
Mattsson, J., Fornehed, J., Selander, G., Palombini, F., and C. Amsuess, "Controlling Actuators with CoAP", draft-mattsson-core-coap-actuators-06 (work in progress), September 2018.

- [I-D.mcgrew-iv-gen] McGrew, D., "Generation of Deterministic Initialization Vectors (IVs) and Nonces", draft-mcgrew-iv-gen-03 (work in progress), October 2013.
- [MF00] McGrew, D. and S. Fluhrer, "Attacks on Encryption of Redundant Plaintext and Implications on Internet Security", the Proceedings of the Seventh Annual Workshop on Selected Areas in Cryptography (SAC 2000), Springer-Verlag. , 2000.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <<https://www.rfc-editor.org/info/rfc3552>>.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <<https://www.rfc-editor.org/info/rfc5116>>.
- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <<https://www.rfc-editor.org/info/rfc5869>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7515] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", RFC 7515, DOI 10.17487/RFC7515, May 2015, <<https://www.rfc-editor.org/info/rfc7515>>.
- [RFC7967] Bhattacharyya, A., Bandyopadhyay, S., Pal, A., and T. Bose, "Constrained Application Protocol (CoAP) Option for No Server Response", RFC 7967, DOI 10.17487/RFC7967, August 2016, <<https://www.rfc-editor.org/info/rfc7967>>.

Appendix A. Scenario Examples

This section gives examples of OSCORE, targeting scenarios in Section 2.2.1.1 of [I-D.hartke-core-e2e-security-reqs]. The message exchanges are made, based on the assumption that there is a security context established between client and server. For simplicity, these examples only indicate the content of the messages without going into detail of the (compressed) COSE message format.

A.1. Secure Access to Sensor

This example illustrates a client requesting the alarm status from a server.

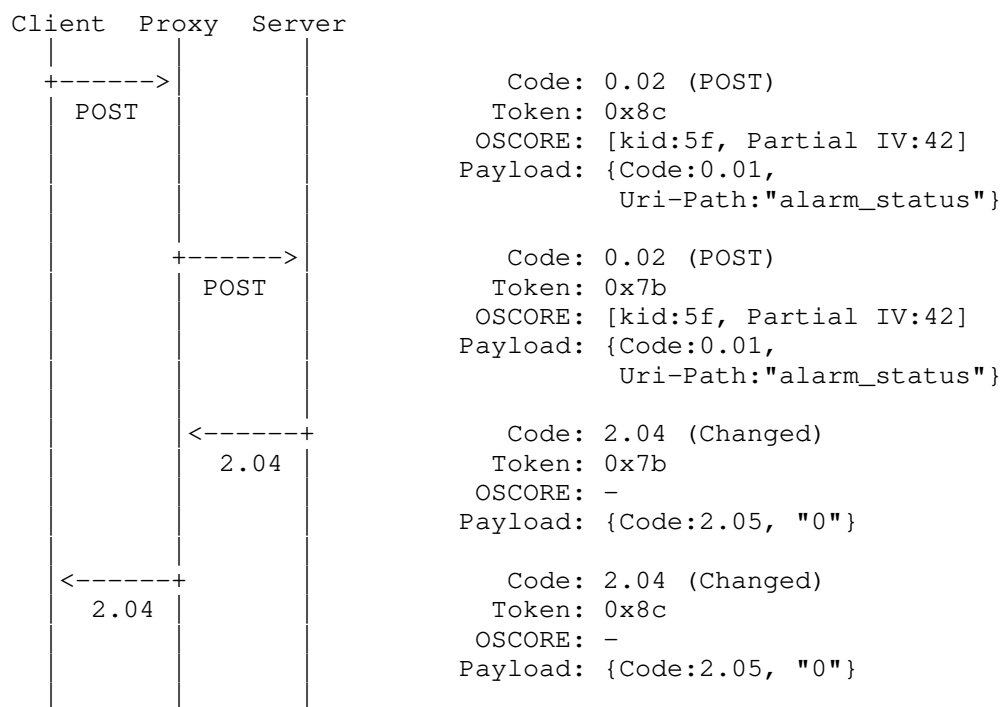


Figure 12: Secure Access to Sensor. Square brackets [...] indicate content of compressed COSE object. Curly brackets { ... } indicate encrypted data.

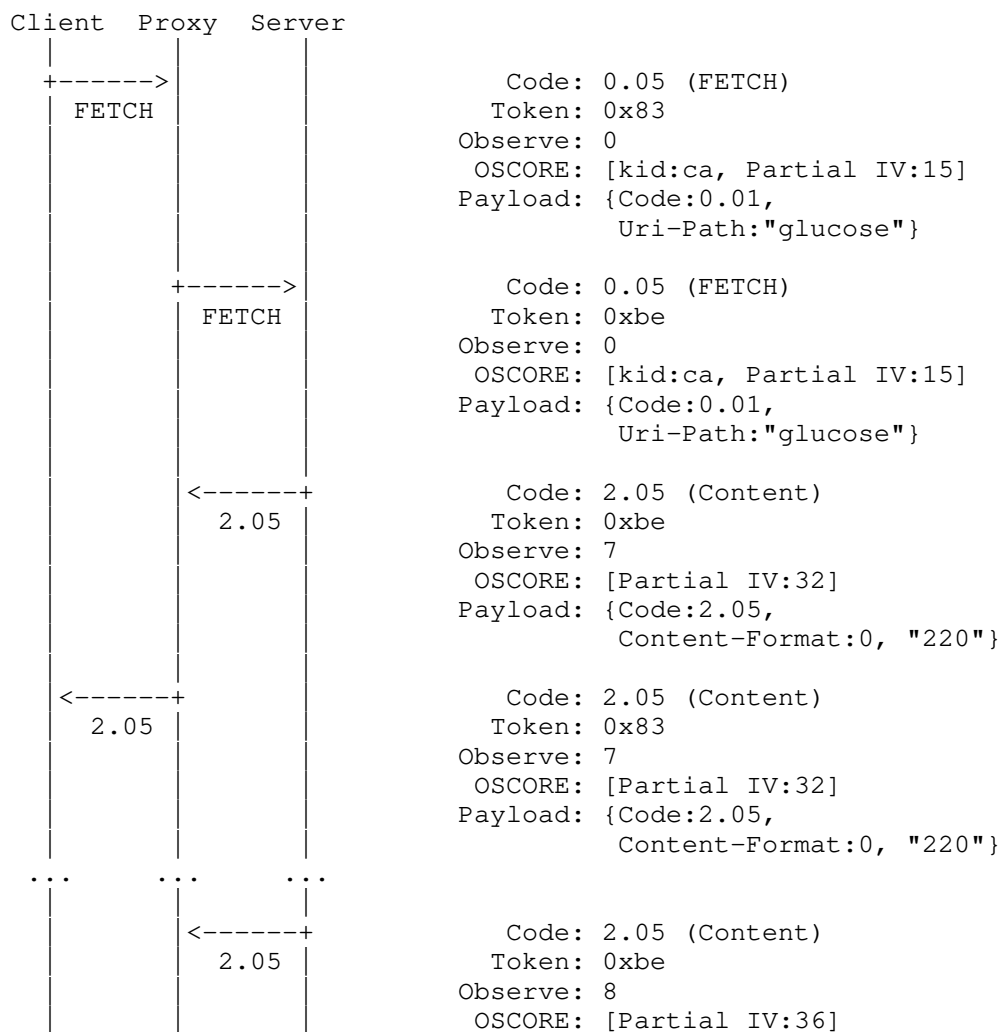
The request/response Codes are encrypted by OSCORE and only dummy Codes (POST/Changed) are visible in the header of the OSCORE message. The option Uri-Path ("alarm_status") and payload ("0") are encrypted.

The COSE header of the request contains an identifier (5f), indicating which security context was used to protect the message and a Partial IV (42).

The server verifies the request as specified in Section 8.2. The client verifies the response as specified in Section 8.4.

A.2. Secure Subscribe to Sensor

This example illustrates a client requesting subscription to a blood sugar measurement resource (GET /glucose), first receiving the value 220 mg/dl and then a second value 180 mg/dl.



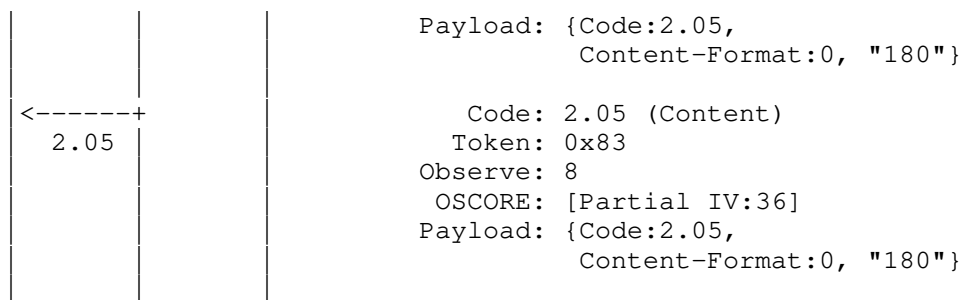


Figure 13: Secure Subscribe to Sensor. Square brackets [...] indicate content of compressed COSE object header. Curly brackets { ... } indicate encrypted data.

The dummy Codes (FETCH/Content) are used to allow forwarding of Observe messages. The options Content-Format (0) and the payload ("220" and "180"), are encrypted.

The COSE header of the request contains an identifier (ca), indicating the security context used to protect the message and a Partial IV (15). The COSE headers of the responses contains Partial IVs (32 and 36).

The server verifies that the Partial IV has not been received before. The client verifies that the responses are bound to the request and that the Partial IVs are greater than any Partial IV previously received in a response bound to the request.

Appendix B. Deployment Examples

For many IoT deployments, a 128 bit uniformly random Master Key is sufficient for encrypting all data exchanged with the IoT device throughout its lifetime. Two examples are given in this section. In the first example, the security context is only derived once from the Master Secret. In the second example, security contexts are derived multiple times using random inputs.

B.1. Security Context Derived Once

An application that only derives the security context once needs to handle the loss of mutable security context parameters, e.g. due to reboot.

B.1.1.1. Sender Sequence Number

In order to handle loss of Sender Sequence Numbers, the device may implement procedures for writing to non-volatile memory during normal operations and updating the security context after reboot, provided that the procedures comply with the requirements on the security context parameters (Section 3.3). This section gives an example of such a procedure.

There are known issues related to writing to non-volatile memory. For example, flash drives may have a limited number of erase operations during its life time. Also, the time for a write operation to non-volatile memory to be completed may be unpredictable, e.g. due to caching, which could result in important security context data not being stored at the time when the device reboots.

However, many devices have predictable limits for writing to non-volatile memory, are physically limited to only send a small amount of messages per minute, and may have no good source of randomness.

To prevent reuse of Sender Sequence Numbers (SSN), an endpoint may perform the following procedure during normal operations:

- o Before using a Sender Sequence Number that is evenly divisible by K, where K is a positive integer, store the Sender Sequence Number (SSN1) in non-volatile memory. After boot, the endpoint initiates the new Sender Sequence Number (SSN2) to the value stored in persistent memory plus K plus F: $SSN2 = SSN1 + K + F$, where F is a positive integer.
 - * Writing to non-volatile memory can be costly; the value K gives a trade-off between frequency of storage operations and efficient use of Sender Sequence Numbers.
 - * Writing to non-volatile memory may be subject to delays, or failure; F MUST be set so that the last Sender Sequence Number used before reboot is never larger than SSN2.

If F cannot be set so SSN2 is always larger than the last Sender Sequence Number used before reboot, the method described in this section MUST NOT be used.

B.1.1.2. Replay Window

In case of loss of security context on the server, to prevent accepting replay of previously received requests, the server may perform the following procedure after boot:

- o The server updates its Sender Sequence Number as specified in Appendix B.1.1, to be used as Partial IV in the response containing the Echo option (next bullet).
- o For each stored security context, the first time after boot the server receives an OSCORE request, the server responds with an OSCORE protected 4.01 (Unauthorized), containing only the Echo option [I-D.ietf-core-echo-request-tag] and no diagnostic payload. The server MUST use its Partial IV when generating the AEAD nonce and MUST include the Partial IV in the response (see Section 5). If the server with use of the Echo option can verify a second OSCORE request as fresh, then the Partial IV of the second request is set as the lower limit of the replay window of that security context.

B.1.3. Notifications

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

- o The client forgets about earlier registrations, removes all Notification Numbers and registers using Observe.

B.2. Security Context Derived Multiple Times

An application which does not require forward secrecy may allow multiple security contexts to be derived from one Master Secret. The requirements on the security context parameters MUST be fulfilled (Section 3.3) even if the client or server is rebooted, recommissioned or in error cases.

This section gives an example of a protocol which adds randomness to the ID Context parameter and uses that together with input parameters pre-established between client and server, in particular Master Secret, Master Salt, and Sender/Recipient ID (see Section 3.2), to derive new security contexts. The random input is transported between client and server in the 'kid context' parameter. This protocol MUST NOT be used unless both endpoints have good sources of randomness.

During normal requests the ID Context of an established security context may be sent in the 'kid context' which, together with 'kid', facilitates for the server to locate a security context. Alternatively, the 'kid context' may be omitted since the ID Context is expected to be known to both client and server, see Section 5.1.

The protocol described in this section may only be needed when the mutable part of security context is lost in the client or server,

e.g. when the endpoint has rebooted. The protocol may additionally be used whenever the client and server need to derive a new security context. For example, if a device is provisioned with one fixed set of input parameters (including Master Secret, Sender and Recipient Identifiers) then a randomized ID Context ensures that the security context is different for each deployment.

The protocol is described below with reference to Figure 14. The client or the server may initiate the protocol, in the latter case step 1 is omitted.

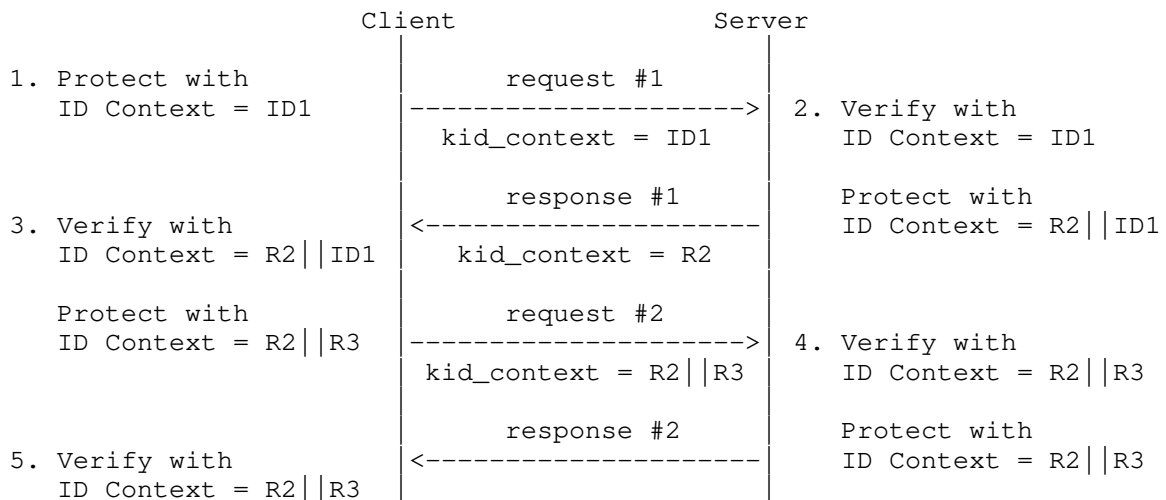


Figure 14: Protocol for establishing a new security context.

1. (Optional) If the client does not have a valid security context with the server, e.g. because of reboot or because this is the first time it contacts the server, then it generates a random string R1, and uses this as ID Context together with the input parameters shared with the server to derive a first security context. The client sends an OSCORE request to the server protected with the first security context, containing R1 wrapped in a CBOR bstr as 'kid context'. The request may target a special resource used for updating security contexts.
2. The server receives an OSCORE request for which it does not have a valid security context, either because the client has generated a new security context ID1 = R1, or because the server has lost part of its security context, e.g. ID Context, Sender Sequence Number or replay window. If the server is able to verify the request (see Section 8.2) with the new derived first security context using the received ID1 (transported in 'kid context') as

ID Context and the input parameters associated to the received 'kid', then the server generates a random string R2, and derives a second security context with ID Context = ID2 = R2 || ID1. The server sends a 4.01 (Unauthorized) response protected with the second security context, containing R2 wrapped in a CBOR bstr as 'kid context', and caches R2. R2 MUST NOT be reused as that may lead to reuse of key and nonce in response #1. Note that the server may receive several requests #1 associated with one security context, leading to multiple parallel protocol runs. Multiple instances of R2 may need to be cached until one of the protocol runs is completed, see Appendix B.2.1.

3. The client receives a response with 'kid context' containing a CBOR bstr wrapping R2 to an OSCORE request it made with ID Context = ID1. The client derives a second security context using ID Context = ID2 = R2 || ID1. If the client can verify the response (see Section 8.4) using the second security context, then the client makes a request protected with a third security context derived from ID Context = ID3 = R2 || R3, where R3 is a random byte string generated by the client. The request includes R2 || R3 wrapped in a CBOR bstr as 'kid context'.
4. If the server receives a request with 'kid context' containing a CBOR bstr wrapping ID3, where the first part of ID3 is identical to an R2 sent in a previous response #1 which it has not received before, then the server derives a third security context with ID Context = ID3. The server MUST NOT accept replayed request #2 messages. If the server can verify the request (see Section 8.2) with the third security context, then the server marks the third security context to be used with this client and removes all instances of R2 associated to this security context from the cache. This security context replaces the previous security context with the client, and the first and the second security contexts are deleted. The server responds using the same security context as in the request.
5. If the client receives a response to the request with the third security context and the response verifies (see Section 8.4), then the client marks the third security context to be used with this server. This security context replaces the previous security context with the server, and the first and second security contexts are deleted.

If verification fails in any step, the endpoint stops processing that message.

The length of the nonces R1, R2, and R3 is application specific. The application needs to set the length of each nonce such the

probability of its value being repeated is negligible; typically, at least 8 bytes long. Since R2 may be generated as the result of a replayed request #1, the probability for collision of R2s is impacted by the birthday paradox. For example, setting the length of R2 to 8 bytes results in an average collision after 2^{32} response #1 messages, which should not be an issue for a constrained server handling on the order of one request per second.

Request #2 can be an ordinary request. The server performs the action of the request and sends response #2 after having successfully completed the security context related operations in step 4. The client acts on response #2 after having successfully completed step 5.

When sending request #2, the client is assured that the Sender Key (derived with the random value R3) has never been used before. When receiving response #2, the client is assured that the response (protected with a key derived from the random value R3 and the Master Secret) was created by the server in response to request #2.

Similarly, when receiving request #2, the server is assured that the request (protected with a key derived from the random value R2 and the Master Secret) was created by the client in response to response #1. When sending response #2, the server is assured that the Sender Key (derived with the random value R2) has never been used before.

Implementation and denial-of-service considerations are made in Appendix B.2.1 and Appendix B.2.2.

B.2.1. Implementation Considerations

This section add some implemention considerations to the protocol described in the previous section.

The server may only have space for a few security contexts, or only be able to handle a few protocol runs in parallel. The server may legitimately receive multiple request #1 messages using the same non-mutable security context, e.g. due to packet loss. Replays of old request #1 messages could be difficult for the server to distinguish from legitimate. The server needs to handle the case when the maximum number of cached R2s is reached. If the server receives a request #1 and is not capable of executing it then it may respond with an unprotected 5.03 (Service Unavailable). The server may clear up state from protocol runs which never complete, e.g. set a timer when caching R2, and remove R2 and the associated security contexts from the cache at timeout. Additionally, state information can be flushed at reboot.

As an alternative to caching R2, the server could generate R2 in such a way that it can be sent (in response #1) and verified (at reception of request #2) as the value of R2 it had generated. Such a procedure MUST NOT lead to the server accepting replayed request #2 messages. One construction described in the following is based on using a secret random HMAC key K_HMAC per set of non-mutable security context parameters associated to a client. This construction allows the server to handle verification of R2 in response #2 at the cost of storing the K_HMAC keys and a slightly larger message overhead in response #1. Steps below refer to modifications to Appendix B.2:

- o In step 2, R2 is generated in the following way. First, the server generates a random K_HMAC (unless it already has one associated with the security context), then it sets $R2 = S2 \parallel \text{HMAC}(K_HMAC, S2)$ where S2 is a random byte string, and the HMAC is truncated to 8 bytes. K_HMAC may have an expiration time, after which it is erased. Note that neither R2, S2 nor the derived first and second security contexts need to be cached.
- o In step 4, instead of verifying that R2 coincides with a cached value, the server looks up the associated K_HMAC and verifies the truncated HMAC, and the processing continues accordingly depending on verification success or failure. K_HMAC is used until a run of the protocol is completed (after verification of request #2), or until it expires (whatever comes first), after which K_HMAC is erased. (The latter corresponds to removing the cached values of R2 in step 4 of Appendix B.2, and makes the server reject replays of request #2.)

The length of S2 is application specific and the probability for collision of S2s is impacted by the birthday paradox. For example, setting the length of S2 to 8 bytes results in an average collision after 2^{32} response #1 messages, which should not be an issue for a constrained server handling on the order of one request per second.

Two endpoints sharing a security context may accidentally initiate two instances of the protocol at the same time, each in the role of client, e.g. after a power outage affecting both endpoints. Such a race condition could potentially lead to both protocols failing, and both endpoints repeatedly re-initiating the protocol without converging. Both endpoints can detect this situation and it can be handled in different ways. The requests could potentially be more spread out in time, for example by only initiating this protocol when the endpoint actually needs to make a request, potentially adding a random delay before requests immediately after reboot or if such parallel protocol runs are detected.

B.2.2. Attack Considerations

An on-path attacker may inject a message causing the endpoint to process verification of the message. A message crafted without access to the Master Secret will fail to verify.

Replaying an old request with a value of 'kid_context' which the server does not recognize could trigger the protocol. This causes the server to generate the first and second security context and send a response. But if the client did not expect a response it will be discarded. This may still result in a denial-of-service attack against the server e.g. because of not being able to manage the state associated with many parallel protocol runs, and it may prevent legitimate client requests. Implementation alternatives with less data caching per request #1 message are favorable in this respect, see Appendix B.2.1.

Replaying response #1 in response to some request other than request #1 will fail to verify, since response #1 is associated to request #1, through the dependencies of ID Contexts and the Partial IV of request #1 included in the external_aad of response #1.

If request #2 has already been well received, then the server has a valid security context, so a replay of request #2 is handled by the normal replay protection mechanism. Similarly if response #2 has already been received, a replay of response #2 to some other request from the client will fail by the normal verification of binding of response to request.

Appendix C. Test Vectors

This appendix includes the test vectors for different examples of CoAP messages using OSCORE. Given a set of inputs, OSCORE defines how to set up the Security Context in both the client and the server.

Note that in Appendix C.4 and all following test vectors the Token and the Message ID of the OSCORE-protected CoAP messages are set to the same value of the unprotected CoAP message, to help the reader with comparisons.

[NOTE: the following examples use option number = 9 (TBD1 assigned by IANA). If that differs, the RFC editor is asked to update the test vectors with data provided by the authors. Please remove this paragraph before publication.]

C.1. Test Vector 1: Key Derivation with Master Salt

In this test vector, a Master Salt of 8 bytes is used. The default values are used for AEAD Algorithm and HKDF.

C.1.1. Client

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Master Salt: 0x9e7ca92223786340 (8 bytes)
- o Sender ID: 0x (0 byte)
- o Recipient ID: 0x01 (1 byte)

From the previous parameters,

- o info (for Sender Key): 0x8540f60a634b657910 (9 bytes)
- o info (for Recipient Key): 0x854101f60a634b657910 (10 bytes)
- o info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- o Sender Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- o Recipient Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0x4622d4dd6d944168eefb54987c (13 bytes)
- o recipient nonce: 0x4722d4dd6d944169eefb54987c (13 bytes)

C.1.2. Server

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Master Salt: 0x9e7ca92223786340 (8 bytes)

- o Sender ID: 0x01 (1 byte)
- o Recipient ID: 0x (0 byte)

From the previous parameters,

- o info (for Sender Key): 0x854101f60a634b657910 (10 bytes)
- o info (for Recipient Key): 0x8540f60a634b657910 (9 bytes)
- o info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- o Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o Recipient Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0x4722d4dd6d944169eefb54987c (13 bytes)
- o recipient nonce: 0x4622d4dd6d944168eefb54987c (13 bytes)

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

In this test vector, the default values are used for AEAD Algorithm, HKDF, and Master Salt.

C.2.1. Client

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Sender ID: 0x00 (1 byte)
- o Recipient ID: 0x01 (1 byte)

From the previous parameters,

- o info (for Sender Key): 0x854100f60a634b657910 (10 bytes)
- o info (for Recipient Key): 0x854101f60a634b657910 (10 bytes)

- o info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- o Sender Key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- o Recipient Key: 0xe57b5635815177cd679ab4bcec9d7dda (16 bytes)
- o Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0xbf35ae297d2dace910c52e99f9 (13 bytes)
- o recipient nonce: 0xbf35ae297d2dace810c52e99f9 (13 bytes)

C.2.2. Server

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Sender ID: 0x01 (1 byte)
- o Recipient ID: 0x00 (1 byte)

From the previous parameters,

- o info (for Sender Key): 0x854101f60a634b657910 (10 bytes)
- o info (for Recipient Key): 0x854100f60a634b657910 (10 bytes)
- o info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- o Sender Key: 0xe57b5635815177cd679ab4bcec9d7dda (16 bytes)
- o Recipient Key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- o Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0xbf35ae297d2dace810c52e99f9 (13 bytes)

- o recipient nonce: 0xbf35ae297d2dace910c52e99f9 (13 bytes)

C.3. Test Vector 3: Key Derivation with ID Context

In this test vector, a Master Salt of 8 bytes and a ID Context of 8 bytes are used. The default values are used for AEAD Algorithm and HKDF.

C.3.1. Client

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Master Salt: 0x9e7ca92223786340 (8 bytes)
- o Sender ID: 0x (0 byte)
- o Recipient ID: 0x01 (1 byte)
- o ID Context: 0x37cbf3210017a2d3 (8 bytes)

From the previous parameters,

- o info (for Sender Key): 0x85404837cbf3210017a2d30a634b657910 (17 bytes)
- o info (for Recipient Key): 0x8541014837cbf3210017a2d30a634b657910 (18 bytes)
- o info (for Common IV): 0x85404837cbf3210017a2d30a6249560d (16 bytes)

Outputs:

- o Sender Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- o Recipient Key: 0xe39a0c7c77b43f03b4b39ab9a268699f (16 bytes)
- o Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)
- o recipient nonce: 0x2da58fb85ff1b81d0b7181b85e (13 bytes)

C.3.2. Server

Inputs:

- o Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- o Master Salt: 0x9e7ca92223786340 (8 bytes)
- o Sender ID: 0x01 (1 byte)
- o Recipient ID: 0x (0 byte)
- o ID Context: 0x37cbf3210017a2d3 (8 bytes)

From the previous parameters,

- o info (for Sender Key): 0x8541014837cbf3210017a2d30a634b657910 (18 bytes)
- o info (for Recipient Key): 0x85404837cbf3210017a2d30a634b657910 (17 bytes)
- o info (for Common IV): 0x85404837cbf3210017a2d30a6249560d (16 bytes)

Outputs:

- o Sender Key: 0xe39a0c7c77b43f03b4b39ab9a268699f (16 bytes)
- o Recipient Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- o Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- o sender nonce: 0x2da58fb85ff1b81d0b7181b85e (13 bytes)
- o recipient nonce: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

C.4. Test Vector 4: OSCORE Request, Client

This section contains a test vector for an OSCORE protected CoAP GET request using the security context derived in Appendix C.1. The unprotected request only contains the Uri-Path and Uri-Host options.

Unprotected CoAP request:

0x44015d1f00003974396c6f63616c686f737483747631 (22 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

Sender Context:

- o Sender ID: 0x (0 byte)
- o Sender Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- o Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- o Partial IV: 0x14 (1 byte)
- o kid: 0x (0 byte)
- o external_aad: 0x8501810a40411440 (8 bytes)
- o AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)
- o plaintext: 0x01b3747631 (5 bytes)
- o encryption key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- o nonce: 0x4622d4dd6d944168eefb549868 (13 bytes)

From the previous parameter, the following is derived:

- o OSCORE option value: 0x0914 (2 bytes)
- o ciphertext: 0x612f1092f1776f1c1668b3825e (13 bytes)

From there:

- o Protected CoAP request (OSCORE message): 0x44025d1f00003974396c6f63616c686f7374620914ff612f1092f1776f1c1668b3825e (35 bytes)

C.5. Test Vector 5: OSCORE Request, Client

This section contains a test vector for an OSCORE protected CoAP GET request using the security context derived in Appendix C.2. The unprotected request only contains the Uri-Path and Uri-Host options.

Unprotected CoAP request:
0x440171c30000b932396c6f63616c686f737483747631 (22 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

Sender Context:

- o Sender ID: 0x00 (1 bytes)
- o Sender Key: 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 for an application that sets the ID Context and requires it to be sent in the request, so 'kid context' is present in the protected message. This test vector uses the security context derived in Appendix C.3. The unprotected request only contains the Uri-Path and Uri-Host options.

Unprotected CoAP request:

0x44012f8eef9bbf7a396c6f63616c686f737483747631 (22 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)
- o ID Context: 0x37cbf3210017a2d3 (8 bytes)

Sender Context:

- o Sender ID: 0x (0 bytes)
- o Sender Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- o Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- o Partial IV: 0x14 (1 byte)
- o kid: 0x (0 byte)
- o kid context: 0x37cbf3210017a2d3 (8 bytes)
- o external_aad: 0x8501810a40411440 (8 bytes)
- o AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)
- o plaintext: 0x01b3747631 (5 bytes)
- o encryption key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- o nonce: 0x2ca58fb85ff1b81c0b7181b84a (13 bytes)

From the previous parameter, the following is derived:

- o OSCORE option value: 0x19140837cbf3210017a2d3 (11 bytes)
- o ciphertext: 0x72cd7273fd331ac45cffbe55c3 (13 bytes)

From there:

- o Protected CoAP request (OSCORE message):
0x44022f8eef9bbf7a396c6f63616c686f73746b19140837cbf3210017a2d3ff
72cd7273fd331ac45cffbe55c3 (44 bytes)

C.7. Test Vector 7: OSCORE Response, Server

This section contains a test vector for an OSCORE protected 2.05 (Content) response to the request in Appendix C.4. The unprotected response has payload "Hello World!" and no options. The protected response does not contain a 'kid' nor a Partial IV. Note that some parameters are derived from the request.

Unprotected CoAP response:
0x64455d1f00003974ff48656c6c6f20576f726c6421 (21 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

Sender Context:

- o Sender ID: 0x01 (1 byte)
- o Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o Sender Sequence Number: 0

The following COSE and cryptographic parameters are derived:

- o external_aad: 0x8501810a40411440 (8 bytes)
- o AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)
- o plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)
- o encryption key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)

- o nonce: 0x4622d4dd6d944168eeffb549868 (13 bytes)

From the previous parameter, the following is derived:

- o OSCORE option value: 0x (0 bytes)
- o ciphertext: 0xdbaad1e9a7e7b2a813d3c31524378303cdafae119106 (22 bytes)

From there:

- o Protected CoAP response (OSCORE message):
0x64445d1f0000397490ffdbaad1e9a7e7b2a813d3c31524378303cdafae119106
(32 bytes)

C.8. Test Vector 8: OSCORE Response with Partial IV, Server

This section contains a test vector for an OSCORE protected 2.05 (Content) response to the request in Appendix C.4. The unprotected response has payload "Hello World!" and no options. The protected response does not contain a 'kid', but contains a Partial IV. Note that some parameters are derived from the request.

Unprotected CoAP response:
0x64455d1f00003974ff48656c6c6f20576f726c6421 (21 bytes)

Common Context:

- o AEAD Algorithm: 10 (AES-CCM-16-64-128)
- o Key Derivation Function: HKDF SHA-256
- o Common IV: 0x4622d4dd6d944168eeffb54987c (13 bytes)

Sender Context:

- o Sender ID: 0x01 (1 byte)
- o Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o Sender Sequence Number: 0

The following COSE and cryptographic parameters are derived:

- o Partial IV: 0x00 (1 byte)
- o external_aad: 0x8501810a40411440 (8 bytes)

- o AAD: 0x8368456e63727970743040488501810a40411440 (20 bytes)
- o plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)
- o encryption key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- o nonce: 0x4722d4dd6d944169eefb54987c (13 bytes)

From the previous parameter, the following is derived:

- o OSCORE option value: 0x0100 (2 bytes)
- o ciphertext: 0x4d4c13669384b67354b2b6175ff4b8658c666a6cf88e (22 bytes)

From there:

- o Protected CoAP response (OSCORE message): 0x64445d1f00003974920100ff4d4c13669384b67354b2b6175ff4b8658c666a6cf88e (34 bytes)

Appendix D. Overview of Security Properties

D.1. Threat Model

This section describes the threat model using the terms of [RFC3552].

It is assumed that the endpoints running OSCORE have not themselves been compromised. The attacker is assumed to have control of the CoAP channel over which the endpoints communicate, including intermediary nodes. The attacker is capable of launching any passive or active, on-path or off-path attacks; including eavesdropping, traffic analysis, spoofing, insertion, modification, deletion, delay, replay, man-in-the-middle, and denial-of-service attacks. This means that the attacker can read any CoAP message on the network and undetectably remove, change, or inject forged messages onto the wire.

OSCORE targets the protection of the CoAP request/response layer (Section 2 of [RFC7252]) between the endpoints, including the CoAP Payload, Code, Uri-Path/Uri-Query, and the other Class E option instances (Section 4.1).

OSCORE does not protect the CoAP messaging layer (Section 2 of [RFC7252]) or other lower layers involved in routing and transporting the CoAP requests and responses.

Additionally, OSCORE does not protect Class U option instances (Section 4.1), as these are used to support CoAP forward proxy operations (see Section 5.7.2 of [RFC7252]). The supported proxies

(forwarding, cross-protocol e.g. CoAP to CoAP-mappable protocols such as HTTP) must be able to change certain Class U options (by instruction from the Client), resulting in the CoAP request being redirected to the server. Changes caused by the proxy may result in the request not reaching the server or reaching the wrong server. For cross-protocol proxies, mappings are done on the Outer part of the message so these protocols are essentially used as transport. Manipulation of these options may thus impact whether the protected message reaches or does not reach the destination endpoint.

Attacks on unprotected CoAP message fields generally causes denial-of-service attacks which are out of scope of this document, more details are given in Appendix D.5.

Attacks against the CoAP request-response layer are in scope. OSCORE is intended to protect against eavesdropping, spoofing, insertion, modification, deletion, replay, and man-in-the middle attacks.

OSCORE is susceptible to traffic analysis as discussed later in Appendix D.

D.2. Supporting Proxy Operations

CoAP is designed to work with intermediaries reading and/or changing CoAP message fields to perform supporting operations in constrained environments, e.g. forwarding and cross-protocol translations.

Securing CoAP on transport layer protects the entire message between the endpoints in which case CoAP proxy operations are not possible. In order to enable proxy operations, security on transport layer needs to be terminated at the proxy in which case the CoAP message in its entirety is unprotected in the proxy.

Requirements for CoAP end-to-end security are specified in [I-D.hartke-core-e2e-security-reqs], in particular forwarding is detailed in Section 2.2.1. The client and server are assumed to be honest, while proxies and gateways are only trusted to perform their intended operations.

By working at the CoAP layer, OSCORE enables different CoAP message fields to be protected differently, which allows message fields required for proxy operations to be available to the proxy while message fields intended for the other endpoint remain protected. In the remainder of this section we analyze how OSCORE protects the protected message fields and the consequences of message fields intended for proxy operation being unprotected.

D.3. Protected Message Fields

Protected message fields are included in the Plaintext (Section 5.3) and the Additional Authenticated Data (Section 5.4) of the COSE_Encrypt0 object and encrypted using an AEAD algorithm.

OSCORE depends on a pre-established random Master Secret (Section 12.3) used to derive encryption keys, and a construction for making (key, nonce) pairs unique (Appendix D.4). Assuming this is true, and the keys are used for no more data than indicated in Section 7.2.1, OSCORE should provide the following guarantees:

- o Confidentiality: An attacker should not be able to determine the plaintext contents of a given OSCORE message or determine that different plaintexts are related (Section 5.3).
- o Integrity: An attacker should not be able to craft a new OSCORE message with protected message fields different from an existing OSCORE message which will be accepted by the receiver.
- o Request-response binding: An attacker should not be able to make a client match a response to the wrong request.
- o Non-replayability: An attacker should not be able to cause the receiver to accept a message which it has previously received and accepted.

In the above, the attacker is anyone except the endpoints, e.g. a compromised intermediary. Informally, OSCORE provides these properties by AEAD-protecting the plaintext with a strong key and uniqueness of (key, nonce) pairs. AEAD encryption [RFC5116] provides confidentiality and integrity for the data. Response-request binding is provided by including the 'kid' and Partial IV of the request in the AAD of the response. Non-replayability of requests and notifications is provided by using unique (key, nonce) pairs and a replay protection mechanism (application dependent, see Section 7.4).

OSCORE is susceptible to a variety of traffic analysis attacks based on observing the length and timing of encrypted packets. OSCORE does not provide any specific defenses against this form of attack but the application may use a padding mechanism to prevent an attacker from directly determine the length of the padding. However, information about padding may still be revealed by side-channel attacks observing differences in timing.

D.4. Uniqueness of (key, nonce)

In this section we show that (key, nonce) pairs are unique as long as the requirements in Sections 3.3 and 7.2.1 are followed.

Fix a Common Context (Section 3.1) and an endpoint, called the encrypting endpoint. An endpoint may alternate between client and server roles, but each endpoint always encrypts with the Sender Key of its Sender Context. Sender Keys are (stochastically) unique since they are derived with HKDF using unique Sender IDs, so messages encrypted by different endpoints use different keys. It remains to prove that the nonces used by the fixed endpoint are unique.

Since the Common IV is fixed, the nonces are determined by a Partial IV (PIV) and the Sender ID of the endpoint generating that Partial IV (ID_PIV). The nonce construction (Section 5.2) with the size of the ID_PIV (S) creates unique nonces for different (ID_PIV, PIV) pairs. There are two cases:

A. For requests, and responses with Partial IV (e.g. Observe notifications):

- o ID_PIV = Sender ID of the encrypting endpoint
- o PIV = current Partial IV of the encrypting endpoint

Since the encrypting endpoint steps the Partial IV for each use, the nonces used in case A are all unique as long as the number of encrypted messages is kept within the required range (Section 7.2.1).

B. For responses without Partial IV (e.g. single response to a request):

- o ID_PIV = Sender ID of the endpoint generating the request
- o PIV = Partial IV of the request

Since the Sender IDs are unique, ID_PIV is different from the Sender ID of the encrypting endpoint. Therefore, the nonces in case B are different compared to nonces in case A, where the encrypting endpoint generated the Partial IV. Since the Partial IV of the request is verified for replay (Section 7.4) associated to this Recipient Context, PIV is unique for this ID_PIV, which makes all nonces in case B distinct.

D.5. Unprotected Message Fields

This sections analyses attacks on message fields which are not protected by OSCORE according to the threat model Appendix D.1.

D.5.1. CoAP Header Fields

- o Version. The CoAP version [RFC7252] is not expected to be sensitive to 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]. CoAP proxies are allowed to read and change Token and Token Length between hops. An eavesdropper reading the Token can match requests to responses which can be used in traffic analysis. In particular this is true for notifications, where multiple responses are matched with one request. Modifications of Token and Token Length by an on-path attacker may become a denial-of-service attack, since it may prevent the client to identify to which request the response belongs or to find the correct information to verify integrity of the response.
- o Code. The Outer CoAP Code of an OSCORE message is POST or FETCH for requests with corresponding response codes. An endpoint receiving the message discards the Outer CoAP Code and uses the Inner CoAP Code instead (see Section 4.2). Hence, modifications from attackers to the Outer Code do not impact the receiving endpoint. However, changing the Outer Code from FETCH to a Code value for a method that does not work with Observe (such as POST) may, depending on proxy implementation since Observe is undefined for several Codes, cause the proxy to not forward notifications, which is a denial-of-service attack. The use of FETCH rather than POST reveals no more than what is revealed by the presence of the Outer Observe option.
- o Type/Message ID. The Type/Message ID fields [RFC7252] reveal information about the UDP transport binding, e.g. an eavesdropper reading the Type or Message ID gain information about how UDP messages are related to each other. CoAP proxies are allowed to change Type and Message ID. These message fields are not present in CoAP over TCP [RFC8323], and does not impact the request/response message. A change of these fields in a UDP hop is a denial-of-service attack. By sending an ACK, an attacker can make the endpoint believe that it does not need to retransmit the

previous message. By sending a RST, an attacker may be able to cancel an observation. By changing a NON to a CON, the attacker can cause the receiving endpoint to ACK messages for which no ACK was requested.

- o Length. This field contains the length of the message [RFC8323] which may be used for traffic analysis. These message fields are not present in CoAP over UDP, and does not impact the request/response message. A change of Length is a denial-of-service attack similar to changing TCP header fields.

D.5.2. CoAP Options

- o Max-Age. The Outer Max-Age is set to zero to avoid unnecessary caching of OSCORE error responses. Changing this value thus may cause unnecessary caching. No additional information is carried with this option.
- o Proxy-Uri/Proxy-Scheme. These options are used in CoAP forward proxy deployments. With OSCORE, the Proxy-Uri option does not contain the Uri-Path/Uri-Query parts of the URI. The other parts of Proxy-Uri cannot be protected because forward proxies need to change them in order to perform their functions. The server can verify what scheme is used in the last hop, but not what was requested by the client or what was used in previous hops.
- o Uri-Host/Uri-Port. In forward proxy deployments, the Uri-Host/Uri-Port may be changed by an adversary, and the application needs to handle the consequences of that (see Section 4.1.3.2). The Uri-Host may either be omitted, reveal information equivalent to that of the IP address or more privacy-sensitive information, which is discouraged.
- o Observe. The Outer Observe option is intended for a proxy to support forwarding of Observe messages, but is ignored by the endpoints since the Inner Observe determines the processing in the endpoints. Since the Partial IV provides absolute ordering of notifications it is not possible for an intermediary to spoof reordering (see Section 4.1.3.5). The absence of Partial IV, since only allowed for the first notification, does not prevent correct ordering of notifications. The size and distributions of notifications over time may reveal information about the content or nature of the notifications. Cancellations (Section 4.1.3.5.1) are not bound to the corresponding registrations in the same way responses are bound to requests in OSCORE (see Appendix D.3), but that does not open up for attacks based on mismatched cancellations, since for cancellations to be accepted, all options

in the decrypted message except for ETag Options MUST be the same (see Section 4.1.3.5).

- o Block1/Block2/Size1/Size2. The Outer Block options enables fragmentation of OSCORE messages in addition to segmentation performed by the Inner Block options. The presence of these options indicates a large message being sent and the message size can be estimated and used for traffic analysis. Manipulating these options is a potential denial-of-service attack, e.g. injection of alleged Block fragments. The specification of a maximum size of message, MAX_UNFRAGMENTED_SIZE (Section 4.1.3.4.2), above which messages will be dropped, is intended as one measure to mitigate this kind of attack.
- o No-Response. The Outer No-Response option is used to support proxy functionality, specifically to avoid error transmissions from proxies to clients, and to avoid bandwidth reduction to servers by proxies applying congestion control when not receiving responses. Modifying or introducing this option is a potential denial-of-service attack against the proxy operations, but since the option has an Inner value its use can be securely agreed between the endpoints. The presence of this option is not expected to reveal any sensitive information about the message exchange.
- o OSCORE. The OSCORE option contains information about the compressed COSE header. Changing this field may cause OSCORE verification to fail.

D.5.3. Error and Signaling Messages

Error messages occurring during CoAP processing are protected end-to-end. Error messages occurring during OSCORE processing are not always possible to protect, e.g. if the receiving endpoint cannot locate the right security context. For this setting, unprotected error messages are allowed as specified to prevent extensive retransmissions. Those error messages can be spoofed or manipulated, which is a potential denial-of-service attack.

This document specifies OPTIONAL error codes and specific diagnostic payloads for OSCORE processing error messages. Such messages might reveal information about how many and which security contexts exist on the server. Servers MAY want to omit the diagnostic payload of error messages, use the same error code for all errors, or avoid responding altogether in case of OSCORE processing errors, if that is a security concern for the application. Moreover, clients MUST NOT rely on the error code or the diagnostic payload to trigger specific

actions, as these errors are unprotected and can be spoofed or manipulated.

Signaling messages used in CoAP over TCP [RFC8323] are intended to be hop-by-hop; spoofing signaling messages can be used as a denial-of-service attack of a TCP connection.

D.5.4. HTTP Message Fields

In contrast to CoAP, where OSCORE does not protect header fields to enable CoAP-CoAP proxy operations, the use of OSCORE with HTTP is restricted to transporting a protected CoAP message over an HTTP hop. Any unprotected HTTP message fields may reveal information about the transport of the OSCORE message and enable various denial-of-service attacks. It is RECOMMENDED to additionally use TLS [RFC8446] for HTTP hops, which enables encryption and integrity protection of headers, but still leaves some information for traffic analysis.

Appendix E. CDDL Summary

Data structure definitions in the present specification employ the CDDL language for conciseness and precision. CDDL is defined in [I-D.ietf-cbor-cddl], which at the time of writing this appendix is in the process of completion. As the document is not yet available for a normative reference, the present appendix defines the small subset of CDDL that is being used in the present specification.

Within the subset being used here, a CDDL rule is of the form "name = type", where "name" is the name given to the "type". A "type" can be one of:

- o a reference to another named type, by giving its name. The predefined named types used in the present specification are: "uint", an unsigned integer (as represented in CBOR by major type 0); "int", an unsigned or negative integer (as represented in CBOR by major type 0 or 1); "bstr", a byte string (as represented in CBOR by major type 2); "tstr", a text string (as represented in CBOR by major type 3);
- o a choice between two types, by giving both types separated by a "/";
- o an array type (as represented in CBOR by major type 4), where the sequence of elements of the array is described by giving a sequence of entries separated by commas ",", and this sequence is enclosed by square brackets "[" and "]". Arrays described by an array description contain elements that correspond one-to-one to the sequence of entries given. Each entry of an array description

is of the form "name : type", where "name" is the name given to the entry and "type" is the type of the array element corresponding to this entry.

Acknowledgments

The following individuals provided input to this document: Christian Amsuess, Tobias Andersson, Carsten Bormann, Joakim Brorsson, Ben Campbell, Esko Dijk, Jaro Fietz, Thomas Fossati, Martin Gunnarsson, Klaus Hartke, Mirja Kuehlewind, Kathleen Moriarty, Eric Rescorla, Michael Richardson, Adam Roach, Jim Schaad, Peter van der Stok, Dave Thaler, Martin Thomson, Marco Tiloca, William Vignat, and Malisa Vucinic.

Ludwig Seitz and Goeran Selander worked on this document as part of the CelticPlus project CyberWI, with funding from Vinnova.

Authors' Addresses

Goeran Selander
Ericsson AB

Email: goran.selander@ericsson.com

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
Internet-Draft
Intended status: Standards Track
Expires: 8 September 2021

C. Amsüss, Ed.
Z. Shelby
ARM
M. Koster
SmartThings
C. Bormann
Universitaet Bremen TZI
P. van der Stok
consultant
7 March 2021

CoRE Resource Directory
draft-ietf-core-resource-directory-28

Abstract

In many IoT applications, direct discovery of resources is not practical due to sleeping nodes, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which contains information about resources held on other servers, allowing lookups to be performed for those resources. The input to an RD is composed of links and the output is composed of links constructed from the information stored in the RD. This document specifies the web interfaces that an RD supports for web servers to discover the RD and to register, maintain, lookup and remove information on resources. Furthermore, new target attributes useful in conjunction with an RD are defined.

Note to Readers

Discussion of this document takes place on the CORE Working Group mailing list (core@ietf.org), which is archived at <https://mailarchive.ietf.org/arch/browse/core/> (<https://mailarchive.ietf.org/arch/browse/core/>).

Source for this draft and an issue tracker can be found at <https://github.com/core-wg/resource-directory> (<https://github.com/core-wg/resource-directory>).

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 8 September 2021.

Copyright Notice

Copyright (c) 2021 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	6
3.1. Principles	6
3.2. Architecture	7
3.3. RD Content Model	8
3.4. Link-local addresses and zone identifiers	12
3.5. Use Case: Cellular M2M	12
3.6. Use Case: Home and Building Automation	13
3.7. Use Case: Link Catalogues	14
4. RD discovery and other interface-independent components	14
4.1. Finding a Resource Directory	15
4.1.1. Resource Directory Address Option (RDAO)	17
4.1.2. Using DNS-SD to discover a Resource Directory	19
4.2. Payload Content Formats	19
4.3. URI Discovery	19
5. Registration	22
5.1. Simple Registration	27
5.2. Third-party registration	29
5.3. Operations on the Registration Resource	30

5.3.1.	Registration Update	30
5.3.2.	Registration Removal	34
5.3.3.	Further operations	34
5.3.4.	Request freshness	35
6.	RD Lookup	37
6.1.	Resource lookup	37
6.2.	Lookup filtering	38
6.3.	Resource lookup examples	40
6.4.	Endpoint lookup	42
7.	Security policies	43
7.1.	Endpoint name	44
7.1.1.	Random endpoint names	44
7.2.	Entered resources	44
7.3.	Link confidentiality	45
7.4.	Segmentation	46
7.5.	First-Come-First-Remembered: A default policy	46
8.	Security Considerations	48
8.1.	Discovery	48
8.2.	Endpoint Identification and Authentication	48
8.3.	Access Control	49
8.4.	Denial of Service Attacks	49
8.5.	Skipping freshness checks	50
9.	IANA Considerations	50
9.1.	Resource Types	50
9.2.	IPv6 ND Resource Directory Address Option	51
9.3.	RD Parameter Registry	51
9.3.1.	Full description of the "Endpoint Type" RD Parameter	54
9.4.	"Endpoint Type" (et=) RD Parameter values	54
9.5.	Multicast Address Registration	55
9.6.	Well-Known URIs	55
9.7.	Service Names and Transport Protocol Port Number Registry	55
10.	Examples	56
10.1.	Lighting Installation	56
10.1.1.	Installation Characteristics	56
10.1.2.	RD entries	57
10.2.	OMA Lightweight M2M (LwM2M)	60
11.	Acknowledgments	61
12.	Changelog	61
13.	References	76
13.1.	Normative References	76
13.2.	Informative References	77
Appendix A.	Groups Registration and Lookup	80
Appendix B.	Web links and the Resource Directory	82
B.1.	A simple example	82
B.1.1.	Resolving the URIs	82
B.1.2.	Interpreting attributes and relations	83

B.2. A slightly more complex example	83
B.3. Enter the Resource Directory	84
B.4. A note on differences between link-format and Link header fields	86
Appendix C. Limited Link Format	86
Authors' Addresses	87

1. Introduction

In the work on Constrained RESTful Environments (CoRE), a REST architecture suitable for constrained nodes (e.g. with limited RAM and ROM [RFC7228]) and networks (e.g. 6LoWPAN [RFC4944]) has been established and is used in Internet-of-Things (IoT) or machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC8288]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [RFC6690]. However, [RFC6690] only describes how to discover resources from the web server that hosts them by querying `"/.well-known/core"`. In many constrained scenarios, direct discovery of resources is not practical due to sleeping nodes, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which contains information about resources held on other servers, allowing lookups to be performed for those resources.

This document specifies the web interfaces that an RD supports for web servers to discover the RD and to register, maintain, lookup and remove information on resources. Furthermore, new target attributes useful in conjunction with an RD are defined. 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 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".

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC3986], [RFC8288] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252]. To describe the REST interfaces defined in this specification, the URI Template format is used [RFC6570].

This specification makes use of the following additional terminology:

resolve against

The expression "a URI-reference is resolved against a base URI" is used to describe the process of [RFC3986] Section 5.2.

Noteworthy corner cases are that if the URI-reference is a (full) URI and resolved against any base URI, that gives the original full URI, and that resolving an empty URI reference gives the base URI without any fragment identifier.

Resource Directory (RD)

A web entity that stores information about web resources and implements the REST interfaces defined in this specification for discovery, for the creation, maintenance and removal of registrations, and for lookup of the registered resources.

Sector

In the context of an RD, a sector is a logical grouping of endpoints.

The abbreviation "d=" is used for the sector in query parameters for compatibility with deployed implementations.

Endpoint

Endpoint (EP) is a term used to describe a web server or client in [RFC7252]. In the context of this specification an endpoint is used to describe a web server that registers resources to the RD. An endpoint is identified by its endpoint name, which is included during registration, and has a unique name within the associated sector of the registration.

Registration Base URI

The Base URI of a Registration is a URI that typically gives scheme and authority information about an Endpoint. The Registration Base URI is provided at registration time, and is used by the RD to resolve relative references of the registration into URIs.

Target

The target of a link is the destination address (URI) of the link. It is sometimes identified with "href=", or displayed as "<target>". Relative targets need resolving with respect to the Base URI (section 5.2 of [RFC3986]).

This use of the term Target is consistent with [RFC8288]'s use of the term.

Context

The context of a link is the source address (URI) of the link, and describes which resource is linked to the target. A link's context is made explicit in serialized links as the "anchor=" attribute.

This use of the term Context is consistent with [RFC8288]'s use of the term.

Directory Resource

A resource in the RD containing registration resources.

Registration Resource

A resource in the RD that contains information about an Endpoint and its links.

Commissioning Tool

Commissioning Tool (CT) is a device that assists during installation events by assigning values to parameters, naming endpoints and groups, or adapting the installation to the needs of the applications.

Registrant-ep

Registrant-ep is the endpoint that is registered into the RD. The registrant-ep can register itself, or a CT registers the registrant-ep.

RDAO

Resource Directory Address Option. A new IPv6 Neighbor Discovery option defined for announcing an RD's address.

3. Architecture and Use Cases

3.1. Principles

The RD is primarily a tool to make discovery operations more efficient than querying /.well-known/core on all connected devices, or across boundaries that would limit those operations.

It provides information about resources hosted by other devices that could otherwise only be obtained by directly querying the /.well-known/core resource on these other devices, either by a unicast request or a multicast request.

Information SHOULD only be stored in the RD if it can be obtained by querying the described device's /.well-known/core resource directly.

Data in the RD can only be provided by the device which hosts those data or a dedicated Commissioning Tool (CT). These CTs act on behalf of endpoints too constrained, or generally unable, to present that information themselves. No other client can modify data in the RD. Changes to the information in the RD do not propagate automatically back to the web servers from where the information originated.

3.2. Architecture

The RD architecture is illustrated in Figure 1. An RD is used as a repository of registrations describing resources hosted on other web servers, also called endpoints (EP). An endpoint is a web server associated with a scheme, IP address and port. A physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain RD registrations, and for endpoints to lookup resources from the RD. An RD can be logically segmented by the use of Sectors.

A mechanism to discover an RD using CoRE Link Format [RFC6690] is defined.

Registrations in the RD are soft state and need to be periodically refreshed.

An endpoint uses specific interfaces to register, update and remove a registration. It is also possible for an RD to fetch Web Links from endpoints and add their contents to its registrations.

At the first registration of an endpoint, a "registration resource" is created, the location of which is returned to the registering endpoint. The registering endpoint uses this registration resource to manage the contents of registrations.

A lookup interface for discovering any of the Web Links stored in the RD is provided using the CoRE Link Format.

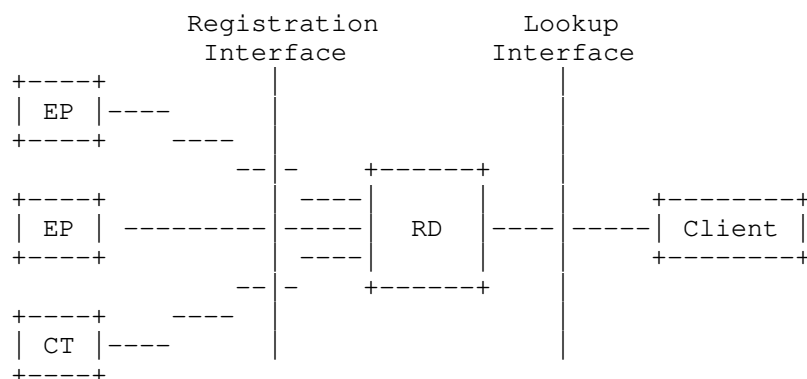


Figure 1: The RD architecture.

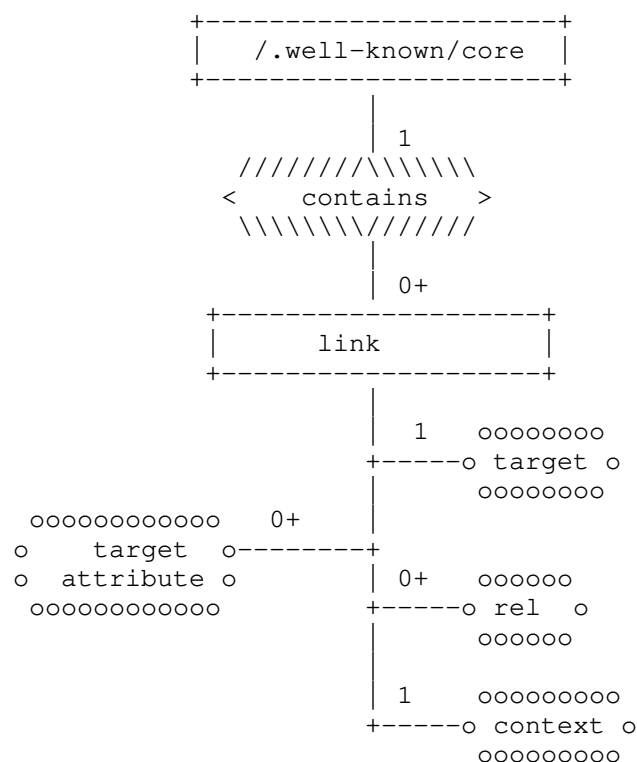
A Registrant-EP MAY keep concurrent registrations to more than one RD at the same time if explicitly configured to do so, but that is not expected to be supported by typical EP implementations. Any such registrations are independent of each other. The usual expectation when multiple discovery mechanisms or addresses are configured is that they constitute a fall-back path for a single registration.

3.3. RD Content Model

The Entity-Relationship (ER) models shown in Figure 2 and Figure 3 model the contents of `/.well-known/core` and the RD 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 an RD. They cover the concepts, but not necessarily all details of an RD's operation; they are meant to give an overview, and not be a template for implementations.

Figure 2: ER Model of the content of `/.well-known/core`

The model shown in Figure 2 models the contents of `/.well-known/core` which contains:

- * a set of links belonging to the hosting web server

The web server is free to choose links it deems appropriate to be exposed in its `/.well-known/core`. Typically, the links describe resources that are served by the host, but the set can also contain links to resources on other servers (see examples in [RFC6690] page 14). The set does not necessarily contain links to all resources served by the host.

A link has the following attributes (see [RFC8288]):

- * **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`.

- * A link context URI: It defines the source of the relation, e.g. `_who_ "hosts" something`.

In link-format serialization, it is expressed in the "anchor" attribute and defaults to the Origin of the target (practically: the target with its path and later components removed)

- * A link target URI: It defines the destination of the relation (e.g. `_what_ is hosted`), and is the topic of all target attributes.

In link-format serialization, it is expressed between angular brackets, and sometimes called the "href".

- * Other target attributes (e.g. resource type (rt), interface (if), or content format (ct)). These provide additional information about the target URI.

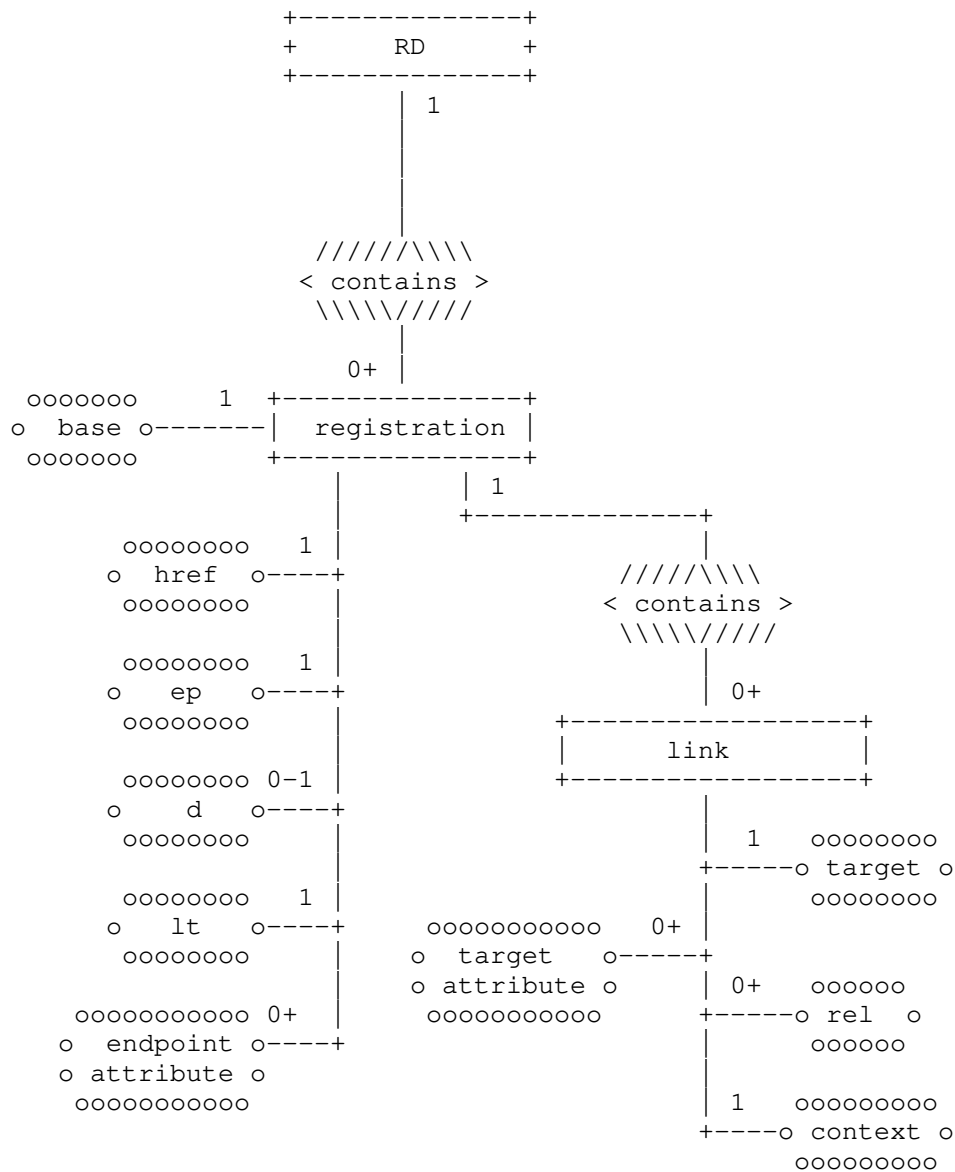


Figure 3: ER Model of the content of the RD

The model shown in Figure 3 models the contents of the RD which contains in addition to /.well-known/core:

* 0 to n Registrations of endpoints,

A registration is associated with one endpoint. A registration defines a set of links as defined for `/.well-known/core`. A Registration has six types of attributes:

- * an endpoint name ("ep", a Unicode string) unique within a sector
- * a Registration Base URI ("base", a URI typically describing the `scheme://authority` part)
- * a lifetime ("lt"),
- * a registration resource location inside the RD ("href"),
- * optionally a sector ("d", a Unicode string)
- * optional additional endpoint attributes (from Section 9.3)

The cardinality of "base" is currently 1; future documents are invited to extend the RD specification to support multiple values (e.g. `[I-D.silverajan-core-coap-protocol-negotiation]`). Its value is used as a Base URI when resolving URIs in the links contained in the endpoint.

Links are modelled as they are in Figure 2.

3.4. Link-local addresses and zone identifiers

Registration Base URIs can contain link-local IP addresses. To be usable across hosts, those cannot be serialized to contain zone identifiers (see `[RFC6874]` Section 1).

Link-local addresses can only be used on a single link (therefore RD servers cannot announce them when queried on a different link), and lookup clients using them need to keep track of which interface they got them from.

Therefore, it is advisable in many scenarios to use addresses with larger scope if available.

3.5. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users: machines. The machines are connected directly to a mobile network using an appropriate embedded wireless interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. The ambition in such systems is to build them from reusable components. These speed up

development and deployment, and enable shared use of machines across different applications. One crucial component of such systems is the discovery of resources (and thus the endpoints they are hosted on) 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 an RD, which is hosted by the mobile operator or somewhere in the cloud. Periodically, these endpoints update their registration and may modify resources they offer.

When endpoints are not always connected, for example because they enter a sleep mode, a remote server is usually used to provide proxy access to the endpoints. Mobile apps or web applications for environment monitoring contact the RD, look up the endpoints capable of providing information about the environment using an appropriate set of link parameters, obtain information on how to contact them (URLs of the proxy server), and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide the appropriate link parameters to the RD to look up for EPs deployed on the vehicles the application is responsible for.

3.6. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the use of IoT 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. Both can use the common RD infrastructure to establish device interactions efficiently, but can pick security policies suitable for their needs.

Two phases can be discerned for a network servicing the system: (1) installation and (2) operation. During the operational phase, the network is connected to the Internet with a Border Router (e.g. a 6LoWPAN Border Router (6LBR), see [RFC6775]) and the nodes connected to the network can use the Internet services that are provided by the Internet Provider or the network administrator. During the installation phase, the network is completely stand-alone, no Border Router is connected, and the network only supports the IP communication between the connected nodes. The installation phase is

usually followed by the operational phase. As an RD's operations work without hard dependencies on names or addresses, it can be used for discovery across both phases.

3.7. Use Case: Link Catalogues

Resources may be shared through data brokers that have no knowledge beforehand of who is going to consume the data. An RD can be used to hold links about resources and services hosted anywhere to make them discoverable by a general class of applications.

For example, environmental and weather sensors that generate data for public consumption may provide data to an intermediary server, or broker. Sensor data are published to the intermediary upon changes or at regular intervals. Descriptions of the sensors that resolve to links to sensor data may be published to an RD. Applications wishing to consume the data can use RD Lookup to discover and resolve links to the desired resources and endpoints. The RD service need not be coupled with the data intermediary service. Mapping of RDs to data intermediaries may be many-to-many.

Metadata in web link formats like [RFC6690] which may be internally stored as triples, or relation/attribute pairs providing metadata about resource links, need to be supported by RDs. External catalogues that are represented in other formats may be converted to common web linking formats for storage and access by RDs. Since it is common practice for these to be encoded in URNs [RFC8141], simple and lossless structural transforms should generally be sufficient to store external metadata in RDs.

The additional features of an RD allow sectors to be defined to enable access to a particular set of resources from particular applications. This provides isolation and protection of sensitive data when needed. Application groups with multicast addresses may be defined to support efficient data transport.

4. RD discovery and other interface-independent components

This and the following sections define the required set of REST interfaces between an RD, endpoints and lookup clients. Although the examples throughout these sections assume the use of CoAP [RFC7252], these REST interfaces can also be realized using HTTP [RFC7230]. The multicast discovery and simple registration operations are exceptions to that, as they rely on mechanisms unavailable in HTTP. In all definitions in these sections, both CoAP response codes (with dot notation) and HTTP response codes (without dot notation) are shown. An RD implementing this specification MUST support the discovery, registration, update, lookup, and removal interfaces.

All operations on the contents of the RD MUST be atomic and idempotent.

For several operations, interface templates are given in list form; those describe the operation participants, request codes, URIs, content formats and outcomes. Sections of those templates contain normative content about Interaction, Method, URI Template and URI Template Variables as well as the details of the Success condition. The additional sections on options like Content-Format and on Failure codes give typical cases that an implementation of the RD should deal with. Those serve to illustrate the typical responses to readers who are not yet familiar with all the details of CoAP based interfaces; they do not limit what a server may respond under atypical circumstances.

REST clients (registrant-EPs and CTs during registration and maintenance, lookup clients, RD servers during simple registrations) must be prepared to receive any unsuccessful code and act upon it according to its definition, options and/or payload to the best of their capabilities, falling back to failing the operation if recovery is not possible. In particular, they SHOULD retry the request upon 5.03 (Service Unavailable; 503 in HTTP) according to the Max-Age (Retry-After in HTTP) option, and SHOULD fall back to link-format when receiving 4.15 (Unsupported Content-Format; 415 in HTTP).

An RD MAY make the information submitted to it available to further directories (subject to security policies on link confidentiality), if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

4.1. Finding a Resource Directory

A (re-)starting device may want to find one or more RDs before it can discover their URIs. Dependent on the operational conditions, one or more of the techniques below apply.

The device may be pre-configured to exercise specific mechanisms for finding the RD:

1. It may be configured with a specific IP address for the RD. That IP address may also be an anycast address, allowing the network to forward RD requests to an RD that is topologically close; each target network environment in which some of these preconfigured nodes are to be brought up is then configured with a route for this anycast address that leads to an appropriate RD. (Instead of using an anycast address, a multicast address can also be preconfigured. The RD servers then need to configure one of their interfaces with this multicast address.)
2. It may be configured with a DNS name for the RD and use DNS to return the IP address of the RD; it can find a DNS server to perform the lookup using the usual mechanisms for finding DNS servers.
3. It may be configured to use a service discovery mechanism such as DNS-SD, as outlined in Section 4.1.2.

For cases where the device is not specifically configured with a way to find an RD, the network may want to provide a suitable default.

1. The IPv6 Neighbor Discovery option RDAO Section 4.1.1 can do that.
2. When DHCP is in use, 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 RD.

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 an RD (using the ABRO option to find that [RFC6775]). Confirmation can be obtained by sending a unicast to "coap://[6LBR]/.well-known/core?rt=core.rd*".
2. In a network that supports multicast well, discovering the RD using a multicast query for /.well-known/core as specified in CoRE Link Format [RFC6690]: Sending a Multicast GET to "coap://[MCD1]/.well-known/core?rt=core.rd*". RDs within the multicast scope will answer the query.

When answering a multicast request directed at a link-local group, the RD may want to respond from a routable address; this makes it easier for registrants to use one of their own routable addresses for

registration. When [RFC6724] is used for source address selection, this can be achieved by applying the changes of its Section 10.4, picking public addresses in its Section 5 Rule 7, and superseding rule 8 with preferring the source address's precedence.

As some of the RD addresses obtained by the methods listed here are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don't work out. For example, an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.

The following RD discovery mechanisms are recommended:

- * In managed networks with border routers that need stand-alone operation, the RDAO option is recommended (e.g. operational phase described in Section 3.6).
- * In managed networks without border router (no Internet services available), the use of a preconfigured anycast address is recommended (e.g. installation phase described in Section 3.6).
- * In networks managed using DNS-SD, the use of DNS-SD for discovery as described in Section 4.1.2 is recommended.

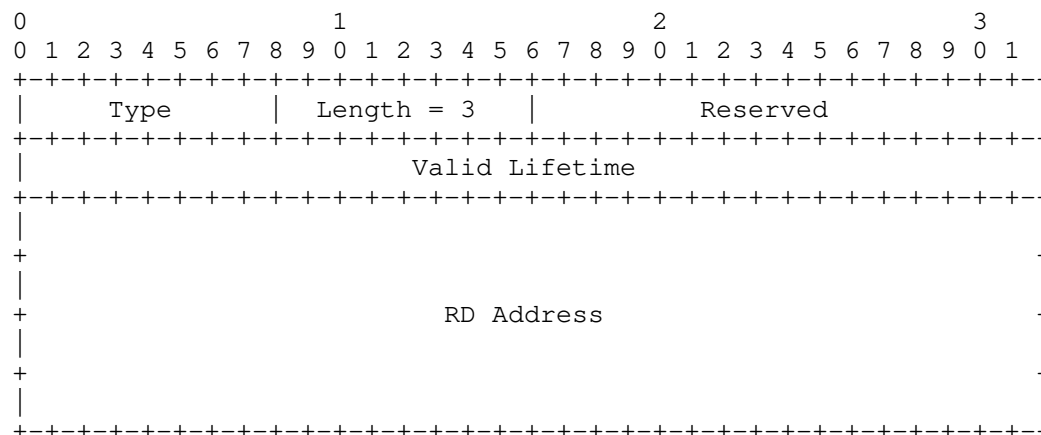
The use of multicast discovery in mesh networks is NOT RECOMMENDED.

4.1.1. Resource Directory Address Option (RDAO)

The Resource Directory Address Option (RDAO) carries information about the address of the RD in RAs (Router Advertisements) of IPv6 Neighbor Discovery (ND), similar to how RDNSS options [RFC8106] are sent. This information is needed when endpoints cannot discover the RD with a link-local or realm-local scope multicast address, for instance because the endpoint and the RD are separated by a Border Router (6LBR). In many circumstances the availability of DHCP cannot be guaranteed either during commissioning of the network. The presence and the use of the RD is essential during commissioning.

It is possible to send multiple RDAO options in one message, indicating as many RD addresses.

The RDAO format is:



Fields:

Type: TBD38

Length: 8-bit unsigned integer. The length of the option in units of 8 bytes. Always 3.

```
Reserved:      This field is unused.  It MUST be
                initialized to zero by the sender and
                MUST be ignored by the receiver.
```

Valid Lifetime: 32-bit unsigned integer. The length of time in seconds (relative to the time the packet is received) that this RD address is valid. A value of all zero bits (0x0) indicates that this RD address is not valid anymore.

RD Address: IPv6 address of the RD.

Figure 4: Resource Directory Address Option

4.1.2. Using DNS-SD to discover a Resource Directory

An RD can advertise its presence in DNS-SD [RFC6763] using the service name "_core-rd._udp" (for CoAP), "_core-rd-dtls._udp" (for CoAP over DTLS), "_core-rd._tcp" (for CoAP over TCP) or "_core-rd-tls._tcp" (for CoAP over TLS) defined in this document. (For the WebSocket transports of CoAP, no service is defined as DNS-SD is typically unavailable in environments where CoAP over WebSockets is used).

The selection of the service indicates the protocol used, and the SRV record points the client to a host name and port to use as a starting point for the URI discovery steps of Section 4.3.

This section is a simplified concrete application of the more generic mechanism specified in [I-D.ietf-core-rd-dns-sd].

4.2. Payload Content Formats

RDs implementing this specification MUST support the application/link-format content format (ct=40).

RDs implementing this specification MAY support additional content formats.

Any additional content format supported by an RD implementing this specification SHOULD be able to express all the information expressible in link-format. It MAY be able to express information that is inexpressible in link-format, but those expressions SHOULD be avoided where possible.

4.3. URI Discovery

Before an endpoint can make use of an RD, it must first know the RD's address and port, and the URI path information for its REST APIs. This section defines discovery of the RD and its URIs using the well-known interface of the CoRE Link Format [RFC6690] after having discovered a host as described in Section 4.1.

Discovery of the RD registration URI is performed by sending either a multicast or unicast GET request to `"/.well-known/core"` and including a Resource Type (rt) parameter [RFC6690] with the value `"core.rd"` in the query string. Likewise, a Resource Type parameter value of `"core.rd-lookup"` is used to discover the URIs for RD Lookup operations, `core.rd*` is used to discover all URIs for RD operations. Upon success, the response will contain a payload with a link format entry for each RD function discovered, indicating the URI of the RD function returned and the corresponding Resource Type. When

performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network (see Section 9.5).

An RD MAY provide hints about the content-formats it supports in the links it exposes or registers, using the "ct" target attribute, as shown in the example below. Clients MAY use these hints to select alternate content-formats for interaction with the RD.

HTTP does not support multicast and consequently only unicast discovery can be supported at the using the HTTP `"/.well-known/core"` resource.

RDs implementing this specification MUST support query filtering for the `rt` parameter as defined in [RFC6690].

While the link targets in this discovery step are often expressed in path-absolute form, this is not a requirement. Clients of the RD SHOULD therefore accept URIs of all schemes they support, both as URIs and relative references, and not limit the set of discovered URIs to those hosted at the address used for URI discovery.

With security policies where the client requires the RD to be authorized to act as an RD, that authorization may be limited to resources on which the authorized RD advertises the adequate resource types. Clients that have obtained links they can not rely on yet can repeat the URI discovery step at the `/.well-known/core` resource of the indicated host to obtain the resource type information from an authorized source.

The URI Discovery operation can yield multiple URIs of a given resource type. The client of the RD can use any of the discovered addresses initially.

The discovery request interface is specified as follows (this is exactly the Well-Known Interface of [RFC6690] Section 4, with the additional requirement that the server MUST support query filtering):

Interaction: EP, CT or Client -> RD

Method: GET

URI Template: `/.well-known/core{?rt}`

URI Template Variables: `rt` := Resource Type. SHOULD contain one of the values `"core.rd"`, `"core.rd-lookup*"`, `"core.rd-lookup-res"`, `"core.rd-lookup-ep"`, or `"core.rd*"`

Accept: absent, application/link-format or any other media type representing web links

The following response is expected on this interface:

Success: 2.05 "Content" or 200 "OK" with an application/link-format or other web link payload containing one or more matching entries for the RD resource.

The following example shows an endpoint discovering an RD using this interface, thus learning that the directory resource location, in this example, is /rd, and that the content-format delivered by the server hosting the resource is application/link-format (ct=40). Note that it is up to the RD to choose its RD locations.

Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content

Payload:

```
</rd>;rt=core.rd;ct=40,  
</rd-lookup/ep>;rt=core.rd-lookup-ep;ct=40,  
</rd-lookup/res>;rt=core.rd-lookup-res;ct=40
```

Figure 5: Example discovery exchange

The following example shows the way of indicating that a client may request alternate content-formats. The Content-Format code attribute "ct" MAY include a space-separated sequence of Content-Format codes as specified in Section 7.2.1 of [RFC7252], indicating that multiple content-formats are available. The example below shows the required Content-Format 40 (application/link-format) indicated as well as a CBOR and JSON representation from [I-D.ietf-core-links-json] (which have no numeric values assigned yet, so they are shown as TBD64 and TBD504 as in that draft). The RD resource locations /rd, and /rd-lookup are example values. The server in this example also indicates that it is capable of providing observation on resource lookups.

Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content

Payload:

```
</rd>;rt=core.rd;ct="40 65225",  
</rd-lookup/res>;rt=core.rd-lookup-res;ct="40 TBD64 TBD504";obs,  
</rd-lookup/ep>;rt=core.rd-lookup-ep;ct="40 TBD64 TBD504"
```

Figure 6: Example discovery exchange indicating additional content-formats

For maintenance, management and debugging, it can be useful to identify the components that constitute the RD server. The identification can be used to find client-server incompatibilities, supported features, required updates and other aspects. The Well-Known interface described in Section 4 of [RFC6690] can be used to find such data.

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
```

```
Payload:
```

```
<http://software.example.com/shiny-resource-directory/1.0beta1>;  
rel=impl-info
```

Figure 7: Example exchange of obtaining implementation information, using the relation type currently proposed in the work-in-progress document

Note that depending on the particular server's architecture, such a link could be anchored at the RD server's root (as in this example), or at individual RD components. The latter is to be expected when different applications are run on the same server.

5. Registration

After discovering the location of an RD, a registrant-ep or CT MAY register the resources of the registrant-ep using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message payload in the CoRE Link Format [RFC6690] or other representations of web links, along with query parameters indicating the name of the endpoint, and optionally the sector, lifetime and base URI of the registration. It is expected that other specifications will define further parameters (see Section 9.3). The RD then creates a new registration resource in the RD and returns its location. The receiving endpoint MUST use that location when refreshing registrations using this interface. Registration resources in the RD are kept active for the period indicated by the lifetime parameter. The creating endpoint is responsible for refreshing the registration resource within this period using either the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameters ep and d (sector) does not create multiple registration resources.

The following rules apply for a registration request targeting a given (ep, d) value pair:

- * When the (ep, d) value pair of the registration-request is different from any existing registration, a new registration is generated.
- * When the (ep, d) value pair of the registration-request is equal to an existing registration, the content and parameters of the existing registration are replaced with the content of the registration request. Like the later changes to registration resources, security policies (Section 7) usually require such requests to come from the same device.

The posted link-format document can (and typically does) contain relative references both in its link targets and in its anchors, or contain empty anchors. The RD server needs to resolve these references in order to faithfully represent them in lookups. They are resolved against the base URI of the registration, which is provided either explicitly in the "base" parameter or constructed implicitly from the requester's URI as constructed from its network address and scheme.

For media types to which Appendix C applies (i.e. documents in application/link-format), request bodies MUST be expressed in Limited Link Format.

The registration request interface is specified as follows:

Interaction: EP or CT -> RD

Method: POST

URI Template: {+rd}{?ep,d,lt,base,extra-attrs*}

URI Template Variables: rd := RD registration URI (mandatory).
This is the location of the RD, as obtained from discovery.

ep := Endpoint name (mostly mandatory).
The endpoint name is an identifier that MUST be unique within a sector.

As the endpoint name is a Unicode string, it is encoded in UTF-8 (and possibly pct-encoded) during variable expansion (see [RFC6570] Section 3.2.1). The endpoint name MUST NOT contain any character in the inclusive ranges 0-31 or 127-159.

The maximum length of this parameter is 63 UTF-8 encoded bytes.

If the RD is configured to recognize the endpoint to be authorized to use exactly one endpoint name, the RD assigns that name. In that case, giving the endpoint name becomes optional for the client; if the client gives any other endpoint name, it is not authorized to perform the registration.

d := Sector (optional). The sector to which this endpoint belongs. When this parameter is not present, the RD MAY associate the endpoint with a configured default sector (possibly based on the endpoint's authorization) or leave it empty.

The sector is encoded like the ep parameter, and is limited to 63 UTF-8 encoded bytes as well.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 1-4294967295. If no lifetime is included in the initial registration, a default value of 90000 (25 hours) SHOULD be assumed.

base := Base URI (optional). This parameter sets the base URI of the registration, under which the relative links in the payload are to be interpreted. The specified URI typically does not have a path component of its own, and MUST be suitable as a base URI to resolve any relative references given in the registration. The parameter is therefore usually of the shape "scheme://authority" for HTTP and CoAP URIs. The URI SHOULD NOT have a query or fragment component as any non-empty relative part in a reference would remove those parts from the resulting URI.

In the absence of this parameter the scheme of the protocol, source address and source port of the registration request are assumed. The Base URI is consecutively constructed by concatenating the used protocol's scheme with the characters "://", the requester's source address as an address literal and ":" followed by its port (if it was not the protocol's default one) in analogy to [RFC7252] Section 6.5.

This parameter is mandatory when the directory is filled by a third party such as an commissioning tool.

If the registrant-ep uses an ephemeral port to register with, it MUST include the base parameter in the registration to provide a valid network path.

A registrant that cannot be reached by potential lookup clients

at the address it registers from (e.g. because it is behind some form of Network Address Translation (NAT)) MUST provide a reachable base address with its registration.

If the Base URI contains a link-local IP literal, it MUST NOT contain a Zone Identifier, and MUST be local to the link on which the registration request is received.

Endpoints that register with a base that contains a path component cannot efficiently express their registrations in Limited Link Format (Appendix C). Those applications should use different representations of links to which Appendix C is not applicable (e.g. [I-D.hartke-t2trg-coral]).

extra-attrs := Additional registration attributes (optional). The endpoint can pass any parameter registered at Section 9.3 to the directory. If the RD is aware of the parameter's specified semantics, it processes it accordingly. Otherwise, it MUST store the unknown key and its value(s) as an endpoint attribute for further lookup.

Content-Format: application/link-format or any other indicated media type representing web links

The following response is expected on this interface:

Success: 2.01 "Created" or 201 "Created". The Location-Path option or Location header field MUST be included in the response. This location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration resource. The registration resource location thus returned is for the purpose of updating the lifetime of the registration and for maintaining the content of the registered links, including updating and deleting links.

A registration with an already registered ep and d value pair responds with the same success code and location as the original registration; the set of links registered with the endpoint is replaced with the links from the payload.

The location MUST NOT have a query or fragment component, as that could conflict with query parameters during the Registration Update operation. Therefore, the Location-Query option MUST NOT be present in a successful response.

If the registration fails, including request timeouts, or if delays from Service Unavailable responses with Max-Age or Retry-After accumulate to exceed the registrant's configured timeouts, it SHOULD

pick another registration URI from the "URI Discovery" step and if there is only one or the list is exhausted, pick other choices from the "Finding a Resource Directory" step. Care has to be taken to consider the freshness of results obtained earlier, e.g. of the result of a `"/.well-known/core"` response, the lifetime of an RDAO option and of DNS responses. Any rate limits and persistent errors from the "Finding a Resource Directory" step must be considered for the whole registration time, not only for a single operation.

The following example shows a registrant-ep with the name "node1" registering two resources to an RD using this interface. The location `"/rd"` is an example RD location discovered in a request similar to Figure 5.

```
Req: POST coap://rd.example.com/rd?ep=node1
Content-Format: 40
Payload:
</sensors/temp>;rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel=describedby

Res: 2.01 Created
Location-Path: /rd/4521
```

Figure 8: Example registration payload

An RD may optionally support HTTP. Here is an example of almost the same registration operation above, when done using HTTP.

```
Req:
POST /rd?ep=node1&base=http://[2001:db8:1::1] HTTP/1.1
Host: rd.example.com
Content-Type: application/link-format

</sensors/temp>;rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel=describedby

Res:
HTTP/1.1 201 Created
Location: /rd/4521
```

Figure 9: Example registration payload as expressed using HTTP

5.1. Simple Registration

Not all endpoints hosting resources are expected to know how to upload links to an RD as described in Section 5. Instead, simple endpoints can implement the Simple Registration approach described in this section. An RD implementing this specification **MUST** implement Simple Registration. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the registrant-ep makes available the hosted resources that it wants to be discovered, as links on its `"/.well-known/core"` interface as specified in [RFC6690]. The links in that document are subject to the same limitations as the payload of a registration (with respect to Appendix C).

- * The registrant-ep finds one or more addresses of the directory server as described in Section 4.1.
- * The registrant-ep sends (and regularly refreshes with) a POST request to the `"/.well-known/rd"` URI of the directory server of choice. The body of the POST request is empty, and triggers the resource directory server to perform GET requests at the requesting registrant-ep's `/.well-known/core` to obtain the link-format payload to register.

The registrant-ep includes the same registration parameters in the POST request as it would with a regular registration per Section 5. The registration base URI of the registration is taken from the registrant-ep's network address (as is default with regular registrations).

Example request from registrant-EP to RD (unanswered until the next step):

```
Req: POST /.well-known/rd?lt=6000&ep=node1
(No payload)
```

Figure 10: First half example exchange of a simple registration

- * The RD queries the registrant-ep's discovery resource to determine the success of the operation. It **SHOULD** keep a cache of the discovery resource and not query it again as long as it is fresh.

Example request from the RD to the registrant-EP:

```
Req: GET /.well-known/core
Accept: 40
```

```
Res: 2.05 Content
Content-Format: 40
Payload:
</sen/temp>
```

Figure 11: Example exchange of the RD querying the simple endpoint

With this response, the RD would answer the previous step's request:

```
Res: 2.04 Changed
```

Figure 12: Second half example exchange of a simple registration

The sequence of fetching the registration content before sending a successful response was chosen to make responses reliable, and the point about caching was chosen to still allow very constrained registrants. Registrants MUST be able to serve a GET request to `"/.well-known/core"` after having requested registration. Constrained devices MAY regard the initial request as temporarily failed when they need RAM occupied by their own request to serve the RD's GET, and retry later when the RD already has a cached representation of their discovery resources. Then, the RD can reply immediately and the registrant can receive the response.

The simple registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: `/.well-known/rd{?ep,d,lt,extra-attrs*}`

URI Template Variables are as they are for registration in Section 5. The base attribute is not accepted to keep the registration interface simple; that rules out registration over CoAP-over-TCP or HTTP that would need to specify one. For some time during this document's development, the URI template `"/.well-known/core{?ep,...}"` has been in use instead.

The following response is expected on this interface:

```
Success: 2.04 "Changed".
```

For the second interaction triggered by the above, the registrant-ep takes the role of server and the RD the role of client. (Note that this is exactly the Well-Known Interface of [RFC6690] Section 4):

Interaction: RD -> EP

Method: GET

URI Template: /.well-known/core

The following response is expected on this interface:

Success: 2.05 "Content".

When the RD uses any authorization credentials to access the endpoint's discovery resource, or when it is deployed in a location where third parties might reach it but not the endpoint, it SHOULD verify that the apparent registrant-ep intends to register with the given registration parameters before revealing the obtained discovery information to lookup clients. An easy way to do that is to verify the simple registration request's sender address using the Echo option as described in [I-D.ietf-core-echo-request-tag] Section 2.4.

The RD MUST delete registrations created by simple registration after the expiration of their lifetime. Additional operations on the registration resource cannot be executed because no registration location is returned.

5.2. Third-party registration

For some applications, even Simple Registration may be too taxing for some very constrained devices, in particular if the security requirements become too onerous.

In a controlled environment (e.g. building control), the RD can be filled by a third party device, called a Commissioning Tool (CT). The commissioning tool can fill the RD from a database or other means. For that purpose scheme, IP address and port of the URI of the registered device is the value of the "base" parameter of the registration described in Section 5.

It should be noted that the value of the "base" parameter applies to all the links of the registration and has consequences for the anchor value of the individual links as exemplified in Appendix B. An eventual (currently non-existing) "base" attribute of the link is not affected by the value of "base" parameter in the registration.

5.3. Operations on the Registration Resource

This section describes how the registering endpoint can maintain the registrations that it created. The registering endpoint can be the registrant-ep or the CT. The registrations are resources of the RD.

An endpoint should not use this interface for registrations that it did not create. This is usually enforced by security policies, which in general require equivalent credentials for creation of and operations on a registration.

After the initial registration, the registering endpoint retains the returned location of the registration resource for further operations, including refreshing the registration in order to extend the lifetime and "keep-alive" the registration. When the lifetime of the registration has expired, the RD SHOULD NOT respond to discovery queries concerning this endpoint. The RD SHOULD continue to provide access to the registration resource after a registration time-out occurs in order to enable the registering endpoint to eventually refresh the registration. The RD MAY eventually remove the registration resource for the purpose of garbage collection. If the registration resource is removed, the corresponding endpoint will need to be re-registered.

The registration resource may also be used cancel the registration using DELETE, and to perform further operations beyond the scope of this specification.

Operations on the registration resource are sensitive to reordering; Section 5.3.4 describes how order is restored.

The operations on the registration resource are described below.

5.3.1. Registration Update

The update interface is used by the registering endpoint to refresh or update its registration with an RD. To use the interface, the registering endpoint sends a POST request to the registration resource returned by the initial registration operation.

An update MAY update registration parameters like lifetime, base URI or others. 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 are included as query parameters in an update operation as in Section 5.

A registration update resets the timeout of the registration to the (possibly updated) lifetime of the registration, independent of whether a "lt" parameter was given.

If the base URI of the registration is changed in an update, relative references submitted in the original registration or later updates are resolved anew against the new base.

The registration update operation only describes the use of POST with an empty payload. Future standards might describe the semantics of using content formats and payloads with the POST method to update the links of a registration (see Section 5.3.3).

The update registration request interface is specified as follows:

Interaction: EP or CT -> RD

Method: POST

URI Template: {+location}{?lt,base,extra-attrs*}

URI Template Variables: location := This is the Location returned by the RD as a result of a successful earlier registration.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 1-4294967295. If no lifetime is included, the previous last lifetime set on a previous update or the original registration (falling back to 90000) SHOULD be used.

base := Base URI (optional). This parameter updates the Base URI established in the original registration to a new value, and is subject to the same restrictions as in the registration.

If the parameter is set in an update, it is stored by the RD as the new Base URI under which to interpret the relative links present in the payload of the original registration.

If the parameter is not set in the request but was set before, the previous Base URI value is kept unmodified.

If the parameter is not set in the request and was not set before either, the source address and source port of the update request are stored as the Base URI.

extra-attrs := Additional registration

attributes (optional). As with the registration, the RD processes them if it knows their semantics. Otherwise, unknown attributes are stored as endpoint attributes, overriding any previously stored endpoint attributes of the same key.

Note that this default behavior does not allow removing an endpoint attribute in an update. For attributes whose functionality depends on the endpoints' ability to remove them in an update, it can make sense to define a value whose presence is equivalent to the absence of a value. As an alternative, an extension can define different updating rules for their attributes. That necessitates either discovery of whether the RD is aware of that extension, or tolerating the default behavior.

Content-Format: none (no payload)

The following responses are expected on this interface:

Success: 2.04 "Changed" or 204 "No Content" if the update was successfully processed.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have been removed).

If the registration update fails in any way, including "Not Found" and request timeouts, or if the time indicated in a Service Unavailable Max-Age/Retry-After exceeds the remaining lifetime, the registering endpoint SHOULD attempt registration again.

The following example shows how the registering endpoint resets the timeout on its registration resource at an RD using this interface with the example location value: /rd/4521.

Req: POST /rd/4521

Res: 2.04 Changed

Figure 13: Example update of a registration

The following example shows the registering endpoint updating its registration resource at an RD using this interface with the example location value: /rd/4521. The initial registration by the registering endpoint set the following values:

- * endpoint name (ep)=endpoint1
- * lifetime (lt)=500

* Base URI (base)=coap://local-proxy-old.example.com

* payload of Figure 8

The initial state of the RD is reflected in the following request:

Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.05 Content

Payload:

```
<coap://local-proxy-old.example.com/sensors/temp>;
  rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="coap://local-proxy-old.example.com/sensors/temp";
  rel=describedby
```

Figure 14: Example lookup before a change to the base address

The following example shows the registering endpoint changing the Base URI to "coaps://new.example.com:5684":

Req: POST /rd/4521?base=coaps://new.example.com

Res: 2.04 Changed

Figure 15: Example registration update that changes the base address

The consecutive query returns:

Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.05 Content

Payload:

```
<coaps://new.example.com/sensors/temp>;
  rt=temperature-c;if=sensor,
<http://www.example.com/sensors/temp>;
  anchor="coaps://new.example.com/sensors/temp";
  rel=describedby
```

Figure 16: Example lookup after a change to the base address

5.3.2. Registration Removal

Although RD registrations have soft state and will eventually timeout after their lifetime, the registering endpoint SHOULD explicitly remove an entry from the RD if it knows it will no longer be available (for example on shut-down). This is accomplished using a removal interface on the RD by performing a DELETE on the endpoint resource.

The removal request interface is specified as follows:

Interaction: EP or CT -> RD

Method: DELETE

URI Template: {+location}

URI Template Variables: location := This is the Location returned by the RD as a result of a successful earlier registration.

The following responses are expected on this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may already have been removed).

The following examples shows successful removal of the endpoint from the RD with example location value /rd/4521.

Req: DELETE /rd/4521

Res: 2.02 Deleted

Figure 17: Example of a registration removal

5.3.3. Further operations

Additional operations on the registration can be specified in future documents, for example:

- * Send iPATCH (or PATCH) updates ([RFC8132]) to add, remove or change the links of a registration.
- * Use GET to read the currently stored set of links in a registration resource.

Those operations are out of scope of this document, and will require media types suitable for modifying sets of links.

5.3.4. Request freshness

Some security mechanisms usable with an RD allow out of order request processing, or do not even mandate replay protection at all. The RD needs to ensure that operations on the registration resource are executed in an order that does not distort the client's intentions.

This ordering of operations is expressed in terms of freshness as defined in [I-D.ietf-core-echo-request-tag]. Requests that alter a resource's state need to be fresh relative to the latest request that altered that state in a conflicting way.

An RD SHOULD determine a request's freshness, and MUST use the Echo option if it requires request freshness and can not determine the it in any other way. An endpoint MUST support the use of the Echo option. (One reason why an RD would not require freshness is when no relevant registration properties are covered by its security policies.)

5.3.4.1. Efficient use of Echo by an RD

To keep latency and traffic added by the freshness requirements to a minimum, RDs should avoid naive (sufficient but inefficient) freshness criteria.

Some simple mechanisms the RD can employ are:

- * State counter. The RD can keep a monotonous counter that increments whenever a registration changes. For every registration resource, it stores the post-increment value of that resource's last change. Requests altering them need to have at least that value encoded in their Echo option, and are otherwise rejected with a 4.01 Unauthorized and the current counter value as the Echo value. If other applications on the same server use Echo as well, that encoding may include a prefix indicating that it pertains to the RD's counter.

The value associated with a resource needs to be kept across the removal of registrations if the same registration resource is to be reused.

The counter can be reset (and the values of removed resources forgotten) when all previous security associations are reset.

This is the "Persistent Counter" method of
[I-D.ietf-core-echo-request-tag] Appendix A.

- * Preemptive Echo values. The current state counter can be sent in an Echo option not only when requests are rejected with 4.01 Unauthorized, but also with successful responses. Thus, clients can be provided with Echo values sufficient for their next request on a regular basis.

While endpoints may discard received Echo values at leisure between requests, they are encouraged to retain these values for the next request to avoid additional round trips.

- * If the RD can ensure that only one security association has modifying access to any registration at any given time, and that security association provides order on the requests, that order is sufficient to show request freshness.

5.3.4.2. Examples of Echo usage

Figure 18 shows the interactions of an endpoint that has forgotten the server's latest Echo value and temporarily reduces its registration lifetime:

Req: POST /rd/4521?lt=7200

Res: 4.01 Unauthorized
Echo: 0x0123

(EP tries again immediately)

Req: POST /rd/4521?lt=7200
Echo: 0x0123

Res: 2.04 Changed
Echo: 0x0124

(Later the EP regains its confidence in its long-term reachability)

Req: POST /rd/4521?lt=90000
Echo: 0x0124

Res: 2.04 Changed
Echo: 0x0247

Figure 18: Example update of a registration

The other examples do not show Echo options for simplicity, and because they lack the context for any example values to have meaning.

6. RD Lookup

To discover the resources registered with the RD, a lookup interface must be provided. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. JSON or CBOR link format [I-D.ietf-core-links-json]) or using more advanced interfaces (e.g. supporting context or semantic based lookup) on different resources that are discovered independently.

RD Lookup allows lookups for endpoints and resources using attributes defined in this document and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Thus, an endpoint lookup MUST return a list of endpoints and a resource lookup MUST return a list of links to resources.

The lookup type is selected by a URI endpoint, which is indicated by a Resource Type as per Table 1 below:

Lookup Type	Resource Type	Mandatory
Resource	core.rd-lookup-res	Mandatory
Endpoint	core.rd-lookup-ep	Mandatory

Table 1: Lookup Types

6.1. Resource lookup

Resource lookup results in links that are semantically equivalent to the links submitted to the RD by the registrant. The links and link parameters returned by the lookup are equal to the originally submitted ones, except that the target reference is fully resolved, and that the anchor reference is fully resolved if it is present in the lookup result at all.

Links that did not have an anchor attribute in the registration are returned without an anchor attribute. Links of which href or anchor was submitted as a (full) URI are returned with the respective attribute unmodified.

The above rules allow the client to interpret the response as links without any further knowledge of the storage conventions of the RD. The RD MAY replace the registration base URIs with a configured intermediate proxy, e.g. in the case of an HTTP lookup interface for CoAP endpoints.

If the base URI of a registration contains a link-local address, the RD MUST NOT show its links unless the lookup was made from the link on which the registered endpoint can be reached. The RD MUST NOT include zone identifiers in the resolved URIs.

6.2. Lookup filtering

Using the Accept Option, the requester can control whether the returned list is returned in CoRE Link Format ("application/link-format", default) or in alternate content-formats (e.g. from [I-D.ietf-core-links-json]).

Multiple search criteria MAY be included in a lookup. All included criteria MUST match for a link to be returned. The RD 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 "relation-types" (in the link-format ABNF) match if the search value matches any of their values (see Section 4.1 of [RFC6690]; e.g. "?if=tag:example.net,2020:sensor" matches ";if=example.regname tag:example.net,2020:sensor;"). A resource link also matches a search criterion if its endpoint would match the criterion, and vice versa, an endpoint link matches a search criterion if any of its resource links matches it.

Note that "href" is a valid search criterion and matches target references. Like all search criteria, on a resource lookup it can match the target reference of the resource link itself, but also the registration resource of the endpoint that registered it. Queries for resource link targets MUST be in URI form (i.e. not relative references) and are matched against a resolved link target. Queries for endpoints SHOULD be expressed in path-absolute form if possible and MUST be expressed in URI form otherwise; the RD SHOULD recognize either. The "anchor" attribute is usable for resource lookups, and, if queried, MUST be in URI form as well.

Additional query parameters "page" and "count" 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. Unlike block-wise transfer of a complete result set, these parameters ensure that each chunk of results can be interpreted on its own. This simplifies the processing, but can result in duplicate or missed items when coinciding with changes from the registration interface.

Endpoints that are interested in a lookup result repeatedly or continuously can use mechanisms like ETag caching, resource observation ([RFC7641]), or any future mechanism that might allow more efficient observations of collections. These are advertised, detected and used according to their own specifications and can be used with the lookup interface as with any other resource.

When resource observation is used, every time the set of matching links changes, or the content of a matching link changes, the RD sends a notification with the matching link set. The notification contains the successful current response to the given request, especially with respect to representing zero matching links (see "Success" item below).

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: {+type-lookup-location}{?page,count,search*}

URI Template Variables: type-lookup-location := RD Lookup URI for a given lookup type (mandatory). The address is discovered as described in Section 4.3.

search := Search criteria for limiting the number of results (optional).

The search criteria are an associative array, expressed in a form-style query as per the URI template (see [RFC6570] Sections 2.4.2 and 3.2.8)

page := Page (optional). Parameter cannot be used without the count parameter. Results are returned from result set in pages that contain 'count' links starting from index (page * count). Page numbering starts with zero.

count := Count (optional). Number of

results is limited to this parameter value. If the page parameter is also present, the response MUST only include 'count' links starting with the (page * count) link in the result set from the query. If the count parameter is not present, then the response MUST return all matching links in the result set. Link numbering starts with zero.

Accept: absent, application/link-format or any other indicated media type representing web links

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an "application/link-format" or other web link payload containing matching entries for the lookup.

The payload can contain zero links (which is an empty payload in [RFC6690] link format, but could also be "[]" in JSON based formats), indicating that no entities matched the request.

6.3. Resource lookup examples

The examples in this section assume the existence of CoAP hosts with a default CoAP port 61616. HTTP hosts are possible and do not change the nature of the examples.

The following example shows a client performing a resource lookup with the example resource look-up locations discovered in Figure 5:

Req: GET /rd-lookup/res?rt=tag:example.org,2020:temperature

Res: 2.05 Content

Payload:

```
<coap://[2001:db8:3::123]:61616/temp>;  
  rt="tag:example.org,2020:temperature"
```

Figure 19: Example a resource lookup

A client that wants to be notified of new resources as they show up can use observation:

```
Req: GET /rd-lookup/res?rt=tag:example.org,2020: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="tag:example.org,2020:light",
<coap://[2001:db8:3::124]/south>;rt="tag:example.org,2020:light",
<coap://[2001:db8:3::124]/east>;rt="tag:example.org,2020:light"
```

Figure 20: Example an observing resource lookup

The following example shows a client performing a paginated resource lookup

```
Req: GET /rd-lookup/res?page=0&count=5
```

```
Res: 2.05 Content
Payload:
<coap://[2001:db8:3::123]:61616/res/0>;ct=60,
<coap://[2001:db8:3::123]:61616/res/1>;ct=60,
<coap://[2001:db8:3::123]:61616/res/2>;ct=60,
<coap://[2001:db8:3::123]:61616/res/3>;ct=60,
<coap://[2001:db8:3::123]:61616/res/4>;ct=60
```

```
Req: GET /rd-lookup/res?page=1&count=5
```

```
Res: 2.05 Content
Payload:
<coap://[2001:db8:3::123]:61616/res/5>;ct=60,
<coap://[2001:db8:3::123]:61616/res/6>;ct=60,
<coap://[2001:db8:3::123]:61616/res/7>;ct=60,
<coap://[2001:db8:3::123]:61616/res/8>;ct=60,
<coap://[2001:db8:3::123]:61616/res/9>;ct=60
```

Figure 21: Examples of paginated resource lookup

The following example shows a client performing a lookup of all resources of all endpoints of a given endpoint type. It assumes that two endpoints (with endpoint names "sensor1" and "sensor2") have previously registered with their respective addresses "coap://sensor1.example.com" and "coap://sensor2.example.com", and posted the very payload of the 6th response of section 5 of [RFC6690].

It demonstrates how absolute link targets stay unmodified, while relative ones are resolved:

Req: GET /rd-lookup/res?et=tag:example.com,2020:platform

Res: 2.05 Content

Payload:

```
<coap://sensor1.example.com/sensors>;ct=40;title="Sensor Index",
<coap://sensor1.example.com/sensors/temp>;rt=temperature-c;if=sensor,
<coap://sensor1.example.com/sensors/light>;rt=light-lux;if=sensor,
<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",
<coap://sensor2.example.com/sensors/temp>;rt=temperature-c;if=sensor,
<coap://sensor2.example.com/sensors/light>;rt=light-lux;if=sensor,
<http://www.example.com/sensors/t123>;rel=describedby;
  anchor="coap://sensor2.example.com/sensors/temp",
<coap://sensor2.example.com/t>;rel=alternate;
  anchor="coap://sensor2.example.com/sensors/temp"
```

Figure 22: Example of resource lookup from multiple endpoints

6.4. Endpoint lookup

The endpoint lookup returns links to and information about registration resources, which themselves can only be manipulated by the registering endpoint.

Endpoint registration resources are annotated with their endpoint names (ep), sectors (d, if present) and registration base URI (base; reports the registrant-ep's address if no explicit base was given) as well as a constant resource type (rt="core.rd-ep"); the lifetime (lt) is not reported. Additional endpoint attributes are added as target attributes to their endpoint link unless their specification says otherwise.

Links to endpoints SHOULD be presented in path-absolute form or, if required, as (full) URIs. (This ensures that the output conforms to Limited Link Format as described in Appendix C.)

Base addresses that contain link-local addresses MUST NOT include zone identifiers, and such registrations MUST NOT be shown unless the lookup was made from the same link from which the registration was made.

While Endpoint Lookup does expose the registration resources, the RD does not need to make them accessible to clients. Clients SHOULD NOT attempt to dereference or manipulate them.

An RD can report registrations in lookup whose URI scheme and authority differ from the lookup resource's. Lookup clients MUST be prepared to see arbitrary URIs as registration resources in the results and treat them as opaque identifiers; the precise semantics of such links are left to future specifications.

The following example shows a client performing an endpoint lookup limited to endpoints of endpoint type
"tag:example.com,2020:platform":

Req: GET /rd-lookup/ep?et=tag:example.com,2020:platform

Res: 2.05 Content

Payload:

```
</rd/1234>;base="coap://[2001:db8:3::127]:61616";ep=node5;  
  et="tag:example.com,2020:platform";ct=40;rt=core.rd-ep,  
</rd/4521>;base="coap://[2001:db8:3::129]:61616";ep=node7;  
  et="tag:example.com,2020:platform";ct=40;d=floor-3;  
  rt=core.rd-ep
```

Figure 23: Examples of endpoint lookup

7. Security policies

The security policies that are applicable to an RD strongly depend on the application, and are not set out normatively here.

This section provides a list of aspects that applications should consider when describing their use of the RD, without claiming to cover all cases. It is using terminology of [I-D.ietf-ace-oauth-authz], in which the RD acts as the Resource Server (RS), and both registrant-eps and lookup clients act as Clients (C) with support from an Authorization Server (AS), without the intention of ruling out other (e.g. certificate / public-key infrastructure (PKI) based) schemes.

Any, all or none of the below can apply to an application. Which are relevant depends on its protection objectives.

Security policies are set by configuration of the RD, or by choice of the implementation. Lookup clients (and, where relevant, endpoints) can only trust an RD to uphold them if it is authenticated, and authorized to serve as an RD according to the application's requirements.

7.1. Endpoint name

Whenever an RD needs to provide trustworthy results to clients doing endpoint lookup, or resource lookup with filtering on the endpoint name, the RD must ensure that the registrant is authorized to use the given endpoint name. This applies both to registration and later to operations on the registration resource. It is immaterial whether the client is the registrant-ep itself or a CT is doing the registration: The RD cannot tell the difference, and CTs may use authorization credentials authorizing only operations on that particular endpoint name, or a wider range of endpoint names.

It is up to the concrete security policy to describe how endpoint name and sector are transported when certificates are used. For example, it may describe how SubjectAltName dNSName entries are mapped to endpoint and domain names.

7.1.1. Random endpoint names

Conversely, in applications where the RD does not check the endpoint name, the authorized registering endpoint can generate a random number (or string) that identifies the endpoint. The RD should then remember unique properties of the registrant, associate them with the registration for as long as its registration resource is active (which may be longer than the registration's lifetime), and require the same properties for operations on the registration resource.

Registrants that are prepared to pick a different identifier when their initial attempt (or attempts, in the unlikely case of two subsequent collisions) at registration is unauthorized should pick an identifier at least twice as long as the expected number of registrants; registrants without such a recovery options should pick significantly longer endpoint names (e.g. using UUID URNs [RFC4122]).

7.2. Entered resources

When lookup clients expect that certain types of links can only originate from certain endpoints, then the RD needs to apply filtering to the links an endpoint may register.

For example, if clients use an RD to find a server that provides firmware updates, then any registrant that wants to register (or update) links to firmware sources will need to provide suitable credentials to do so, independently of its endpoint name.

Note that the impact of having undesirable links in the RD depends on the application: if the client requires the firmware server to present credentials as a firmware server, a fraudulent link's impact is limited to the client revealing its intention to obtain updates and slowing down the client until it finds a legitimate firmware server; if the client accepts any credentials from the server as long as they fit the provided URI, the impact is larger.

An RD may also require that links are only registered if the registrant is authorized to publish information about the anchor (or even target) of the link. One way to do this is to demand that the registrant present the same credentials as a client that they'd need to present if contacted as a server at the resources' URI, which may include using the address and port that are part of the URI. Such a restriction places severe practical limitations on the links that can be registered.

As above, the impact of undesirable links depends on the extent to which the lookup client relies on the RD. To avoid the limitations, RD applications should consider prescribing that lookup clients only use the discovered information as hints, and describe which pieces of information need to be verified because they impact the application's security. A straightforward way to verify such information is to request it again from an authorized server, typically the one that hosts the target resource. That similar to what happens in Section 4.3 when the URI discovery step is repeated.

7.3. Link confidentiality

When registrants publish information in the RD that is not available to any client that would query the registrant's /.well-known/core interface, or when lookups to that interface are subject to stricter firewalling than lookups to the RD, the RD may need to limit which lookup clients may access the information.

In this case, the endpoint (and not the lookup clients) needs to be careful to check the RD's authorization. The RD needs to check any lookup client's authorization before revealing information directly (in resource lookup) or indirectly (when using it to satisfy a resource lookup search criterion).

7.4. Segmentation

Within a single RD, different security policies can apply.

One example of this are multi-tenant deployments separated by the sector (d) parameter. Some sectors might apply limitations on the endpoint names available, while others use a random identifier approach to endpoint names and place limits on the entered links based on their attributes instead.

Care must be taken in such setups to determine the applicable access control measures to each operation. One easy way to do that is to mandate the use of the sector parameter on all operations, as no credentials are suitable for operations across sector borders anyway.

7.5. First-Come-First-Remembered: A default policy

The First-Come-First-Remembered policy is provided both as a reference example for a security policy definition, and as a policy that implementations may choose to use as default policy in absence of other configuration. It is designed to enable efficient discovery operations even in ad-hoc settings.

Under this policy, the RD accepts registrations for any endpoint name that is not assigned to an active registration resource, and only accepts registration updates from the same endpoint. The policy is minimal in that towards lookup clients it does not make any of the claims of Section 7.2 and Section 7.3, and its claims on Section 7.1 are limited to the lifetime of that endpoint's registration. It does, however, guarantee towards any endpoint that for the duration of its registration, its links will be discoverable on the RD.

When a registration or operation is attempted, the RD MUST determine the client's subject name or public key:

- * If the client's credentials indicate any subject name that is certified by any authority which the RD recognizes (which may be the system's trust anchor store), all such subject names are stored. With CWT or JWT based credentials (as common with ACE), the Subject (sub) claim is stored as a single name, if it exists. With X.509 certificates, the Common Name (CN) and the complete list of SubjectAltName entries are stored. In both cases, the authority that certified the claim is stored along with the subject, as the latter may only be locally unique.

- * Otherwise, if the client proves possession of a private key, the matching public key is stored. This applies both to raw public keys and to the public keys indicated in certificates that failed the above authority check.
- * If neither is present, a reference to the security session itself is stored. With (D)TLS, that is the connection itself, or the session resumption information if available. With OSCORE, that is the security context.

As part of the registration operation, that information is stored along with the registration resource.

The RD MUST accept all registrations whose registration resource is not already active, as long as they are made using a security layer supported by the RD.

Any operation on a registration resource, including registrations that lead to an existing registration resource, MUST be rejected by the RD unless all the stored information is found in the new request's credentials.

Note that even though subject names are compared in this policy, they are never directly compared to endpoint names, and an endpoint can not expect to "own" any particular endpoint name outside of an active registration -- even if a certificate says so. It is an accepted shortcoming of this approach that the endpoint has no indication of whether the RD remembers it by its subject name or public key; recognition by subject happens on a best-effort base (given the RD may not recognize any authority). Clients MUST be prepared to pick a different endpoint name when rejected by the RD initially or after a change in their credentials; picking an endpoint name as per Section 7.1.1 is an easy option for that.

For this policy to be usable without configuration, clients should not set a sector name in their registrations. An RD can set a default sector name for registrations accepted under this policy, which is useful especially in a segmented setup where different policies apply to different sectors. The configuration of such a behavior, as well as any other configuration applicable to such an RD (i.e. the set of recognized authorities) is out of scope for this document.

8. Security Considerations

The security considerations as described in Section 5 of [RFC8288] and Section 6 of [RFC6690] apply. The `"/.well-known/core"` resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [RFC7252].

Access that is limited or affects sensitive data SHOULD be protected, e.g. using (D)TLS or OSCORE ([RFC8613]; which aspects of the RD this affects depends on the security policies of the application (see Section 7).

8.1. Discovery

Most steps in discovery of the RD, and possibly its resources, are not covered by CoAP's security mechanisms. This will not endanger the security properties of the registrations and lookup itself (where the client requires authorization of the RD if it expects any security properties of the operation), but may leak the client's intention to third parties, and allow them to slow down the process.

To mitigate that, clients can retain the RD's address, use secure discovery options like configured addresses, and send queries for RDs in a very general form (`"?rt=core.rd"` rather than `"?rt=core.rd-lookup-ep"`).

8.2. Endpoint Identification and Authentication

An Endpoint (name, sector) pair is unique within the set of endpoints registered by the RD. An Endpoint MUST NOT be identified by its protocol, port or IP address as these may change over the lifetime of an Endpoint.

Every operation performed by an Endpoint on an RD SHOULD be mutually authenticated using Pre-Shared Key, Raw Public Key or Certificate based security.

Consider the following threat: two devices A and B are registered at a single server. Both devices have unique, per-device credentials for use with DTLS to make sure that only parties with authorization to access A or B can do so.

Now, imagine that a malicious device A wants to sabotage the device B. It uses its credentials during the DTLS exchange. Then, it specifies the endpoint name of device B as the name of its own endpoint in device A. If the server does not check whether the identifier provided in the DTLS handshake matches the identifier used at the CoAP layer then it may be inclined to use the endpoint name for looking up what information to provision to the malicious device.

Endpoint authorization needs to be checked on registration and registration resource operations independently of whether there are configured requirements on the credentials for a given endpoint name (and sector; Section 7.1) or whether arbitrary names are accepted (Section 7.1.1).

Simple registration could be used to circumvent address-based access control: An attacker would send a simple registration request with the victim's address as source address, and later look up the victim's /.well-known/core content in the RD. Mitigation for this is recommended in Section 5.1.

The registration resource path is visible to any client that is allowed endpoint lookup, and can be extracted by resource lookup clients as well. The same goes for registration attributes that are shown as target attributes or lookup attributes. The RD needs to consider this in the choice of registration resource paths, and administrators or endpoint in their choice of attributes.

8.3. Access Control

Access control SHOULD be performed separately for the RD registration and Lookup API paths, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the sector, endpoint or resource level.

The precise access controls necessary (and the consequences of failure to enforce them) depend on the protection objectives of the application and the security policies (Section 7) derived from them.

8.4. Denial of Service Attacks

Services that run over UDP unprotected are vulnerable to unknowingly amplify and distribute a DoS attack as UDP does not require return routability check. Since RD lookup responses can be significantly larger than requests, RDs are prone to this.

[RFC7252] describes this at length in its Section 11.3, including some mitigation by using small block sizes in responses. The upcoming [I-D.ietf-core-echo-request-tag] updates that by describing a source address verification mechanism using the Echo option.

[If this document is published together with or after I-D.ietf-core-echo-request-tag, the above paragraph is replaced with the following:

[RFC7252] describes this at length in its Section 11.3, and [I-D.ietf-core-echo-request-tag] (which updates the former) recommends using the Echo option to verify the request's source address.

]

8.5. Skipping freshness checks

When RD based applications are built in which request freshness checks are not performed, these concerns need to be balanced:

- * When alterations to registration attributes are reordered, an attacker may create any combination of attributes ever set, with the attack difficulty determined by the security layer's replay properties.

For example, if Figure 18 were conducted without freshness assurances, an attacker could later reset the lifetime back to 7200. Thus, the device is made unreachable to lookup clients.

- * When registration updates without query parameters (which just serve to restart the lifetime) can be reordered, an attacker can use intercepted messages to give the appearance of the device being alive to the RD.

This is unacceptable when the RD's security policy promises reachability of endpoints (e.g. when disappearing devices would trigger further investigation), but may be acceptable with other policies.

9. IANA Considerations

9.1. Resource Types

IANA is asked to enter the following values into the Resource Type (rt=) Link Target Attribute Values sub-registry of the Constrained Restful Environments (CoRE) Parameters registry defined in [RFC6690]:

Value	Description	Reference
core.rd	Directory resource of an RD	RFCTHIS Section 4.3
core.rd-lookup-res	Resource lookup of an RD	RFCTHIS Section 4.3
core.rd-lookup-ep	Endpoint lookup of an RD	RFCTHIS Section 4.3
core.rd-ep	Endpoint resource of an RD	RFCTHIS Section 6

Table 2

9.2. IPv6 ND Resource Directory Address Option

This document registers one new ND option type under the sub-registry "IPv6 Neighbor Discovery Option Formats" of the "Internet Control Message Protocol version 6 (ICMPv6) Parameters" registry:

- * Resource Directory Address Option (TBD38)

[The RFC editor is asked to replace TBD38 with the assigned number in the document; the value 38 is suggested.]

9.3. RD Parameter Registry

This specification defines a new sub-registry for registration and lookup parameters called "RD Parameters" under "CoRE Parameters". Although this specification defines a basic set of parameters, it is expected that other standards that make use of this interface will define new ones.

Each entry in the registry must include

- * the human readable name of the parameter,
- * the short name as used in query parameters or target attributes,
- * indication of whether it can be passed as a query parameter at registration of endpoints, as a query parameter in lookups, or be expressed as a target attribute,
- * syntax and validity requirements if any,

- * a description,
- * and a link to reference documentation.

The query parameter MUST be both a valid URI query key [RFC3986] and a token as used in [RFC8288].

The description must give details on whether the parameter can be updated, and how it is to be processed in lookups.

The mechanisms around new RD parameters should be designed in such a way that they tolerate RD implementations that are unaware of the parameter and expose any parameter passed at registration or updates on in endpoint lookups. (For example, if a parameter used at registration were to be confidential, the registering endpoint should be instructed to only set that parameter if the RD advertises support for keeping it confidential at the discovery step.)

Initial entries in this sub-registry are as follows:

Full name	Short	Validity	Use	Description
Endpoint Name	ep	Unicode*	RLA	Name of the endpoint
Lifetime	lt	1-4294967295	R	Lifetime of the registration in seconds
Sector	d	Unicode*	RLA	Sector to which this endpoint belongs
Registration Base URI	base	URI	RLA	The scheme, address and port and path at which this server is available
Page	page	Integer	L	Used for pagination
Count	count	Integer	L	Used for pagination
Endpoint Type	et	Section 9.3.1	RLA	Semantic type of the endpoint (see Section 9.4)

Table 3: RD Parameters

(Short: Short name used in query parameters or target attributes.
 Validity: Unicode* = 63 Bytes of UTF-8 encoded Unicode, with no control characters as per Section 5. Use: R = used at registration, L = used at lookup, A = expressed in target attribute.)

The descriptions for the options defined in this document are only summarized here. To which registrations they apply and when they are to be shown is described in the respective sections of this document. All their reference documentation entries point to this document.

The IANA policy for future additions to the sub-registry is "Expert Review" as described in [RFC8126]. The evaluation should consider formal criteria, duplication of functionality (Is the new entry redundant with an existing one?), topical suitability (E.g. is the described property actually a property of the endpoint and not a property of a particular resource, in which case it should go into the payload of the registration and need not be registered?), and the potential for conflict with commonly used target attributes (For

example, "if" could be used as a parameter for conditional registration if it were not to be used in lookup or attributes, but would make a bad parameter for lookup, because a resource lookup with an "if" query parameter could ambiguously filter by the registered endpoint property or the [RFC6690] target attribute).

9.3.1. Full description of the "Endpoint Type" RD Parameter

An endpoint registering at an RD can describe itself with endpoint types, similar to how resources are described with Resource Types in [RFC6690]. An endpoint type is expressed as a string, which can be either a URI or one of the values defined in the Endpoint Type sub-registry. Endpoint types can be passed in the "et" query parameter as part of extra-attrs at the Registration step, are shown on endpoint lookups using the "et" target attribute, and can be filtered for using "et" as a search criterion in resource and endpoint lookup. Multiple endpoint types are given as separate query parameters or link attributes.

Note that Endpoint Type differs from Resource Type in that it uses multiple attributes rather than space separated values. As a result, RDs implementing this specification automatically support correct filtering in the lookup interfaces from the rules for unknown endpoint attributes.

9.4. "Endpoint Type" (et=) RD Parameter values

This specification establishes a new sub-registry under "CoRE Parameters" called '"Endpoint Type" (et=) RD Parameter values'. The registry properties (required policy, requirements, template) are identical to those of the Resource Type parameters in [RFC6690], in short:

The review policy is IETF Review for values starting with "core", and Specification Required for others.

The requirements to be enforced are:

- * The values MUST be related to the purpose described in Section 9.3.1.
- * The registered values MUST conform to the ABNF reg-rel-type definition of [RFC6690] and MUST NOT be a URI.
- * It is recommended to use the period "." character for segmentation.

The registry initially contains one value:

- * "core.rd-group": An application group as described in Appendix A.

9.5. Multicast Address Registration

IANA is asked to assign the following multicast addresses for use by CoAP nodes:

IPv4 -- "all CoRE Resource Directories" address MCD2 (suggestion: 224.0.1.189), from the "IPv4 Multicast Address Space Registry". As the address is used for discovery that may span beyond a single network, it has come from the Internetwork Control Block (224.0.1.x) [RFC5771].

IPv6 -- "all CoRE Resource Directories" address MCD1 (suggestions FF0X::FE), from the "IPv6 Multicast Address Space Registry", in the "Variable Scope Multicast Addresses" space (RFC 3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only.

[The RFC editor is asked to replace MCD1 and MCD2 with the assigned addresses throughout the document.]

9.6. Well-Known URIs

IANA is asked to permanently register the URI suffix "rd" in the "Well-Known URIs" registry. The change controller is the IETF, this document is the reference.

9.7. Service Names and Transport Protocol Port Number Registry

IANA is asked to enter four new items into the Service Names and Transport Protocol Port Number Registry:

- * Service name: "core-rd", Protocol: "udp", Description: "Resource Directory accessed using CoAP"
- * Service name "core-rd-dtls", Protocol: "udp", Description: "Resource Directory accessed using CoAP over DTLS"
- * Service name: "core-rd", Protocol: "tcp", Description: "Resource Directory accessed using CoAP over TCP"
- * Service name "core-rd-tls", Protocol: "tcp", Description: "Resource Directory accessed using CoAP over TLS"

All in common have this document as their reference.

10. Examples

Two examples are presented: a Lighting Installation example in Section 10.1 and a LwM2M example in Section 10.2.

10.1. Lighting Installation

This example shows a simplified lighting installation which makes use of the RD with a CoAP interface to facilitate the installation and start-up of the application code in the lights and sensors. In particular, the example leads to the definition of a group and the enabling of the corresponding multicast address as described in Appendix A. No conclusions must be drawn on the realization of actual installation or naming procedures, because the example only "emphasizes" some of the issues that may influence the use of the RD and does not pretend to be normative.

10.1.1. Installation Characteristics

The example assumes that the installation is managed. That means that a Commissioning Tool (CT) is used to authorize the addition of nodes, name them, and name their services. The CT can be connected to the installation in many ways: the CT can be part of the installation network, connected by WiFi to the installation network, or connected via GPRS link, or other method.

It is assumed that there are two naming authorities for the installation: (1) the network manager that is responsible for the correct operation of the network and the connected interfaces, and (2) the lighting manager that is responsible for the correct functioning of networked lights and sensors. The result is the existence of two naming schemes coming from the two managing entities.

The example installation consists of one presence sensor, and two luminaries, luminary1 and luminary2, each with their own wireless interface. Each luminary contains three lamps: left, right and middle. Each luminary is accessible through one endpoint. For each lamp a resource exists to modify the settings of a lamp in a luminary. The purpose of the installation is that the presence sensor notifies the presence of persons to a group of lamps. The group of lamps consists of: middle and left lamps of luminary1 and right lamp of luminary2.

Before commissioning by the lighting manager, the network is installed and access to the interfaces is proven to work by the network manager.

At the moment of installation, the network under installation is not necessarily connected to the DNS infrastructure. Therefore, SLAAC IPv6 addresses are assigned to CT, RD, luminaries and the sensor. The addresses shown in Table 4 below stand in for these in the following examples.

Name	IPv6 address
luminary1	2001:db8:4::1
luminary2	2001:db8:4::2
Presence sensor	2001:db8:4::3
RD	2001:db8:4::ff

Table 4: Addresses used in the examples

In Section 10.1.2 the use of RD during installation is presented.

10.1.2. RD entries

It is assumed that access to the DNS infrastructure is not always possible during installation. Therefore, the SLAAC addresses are used in this section.

For discovery, the resource types (rt) of the devices are important. The lamps in the luminaries have `rt=tag:example.com,2020:light`, and the presence sensor has `rt=tag:example.com,2020: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 RD are shown in Table 5 below:

Name	endpoint	resource path	resource type
luminary1	lm_R2-4-015_wndw	/light/left	tag:example.com,2020:light
luminary1	lm_R2-4-015_wndw	/light/middle	tag:example.com,2020:light
luminary1	lm_R2-4-015_wndw	/light/right	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/left	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/middle	tag:example.com,2020:light
luminary2	lm_R2-4-015_door	/light/right	tag:example.com,2020:light
Presence sensor	ps_R2-4-015_door	/ps	tag:example.com,2020:p-sensor

Table 5: RD identifiers

It is assumed that the CT has performed RD discovery and has received a response like the one in the Section 4.3 example.

The CT inserts the endpoints of the luminaries and the sensor in the RD using the registration base URI parameter (base) to specify the interface address:

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=lm_R2-4-015_wndw&base=coap://[2001:db8:4::1]&d=R2-4-015
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020: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="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020: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="tag:example.com,2020:p-sensor"

Res: 2.01 Created
Location-Path: /rd/4523
```

Figure 24: Example of registrations a CT enters into an RD

The sector name d=R2-4-015 has been added for an efficient lookup because filtering on "ep" name is more awkward. The same sector name is communicated to the two luminaries and the presence sensor by the CT.

The group is specified in the RD. The base parameter is set to the site-local multicast address allocated to the group. In the POST in the example below, the resources supported by all group members are published.

```
Req: POST coap://[2001:db8:4::ff]/rd
      ?ep=grp_R2-4-015&et=core.rd-group&base=coap://[ff05::1]
Payload:
</light/left>;rt="tag:example.com,2020:light",
</light/middle>;rt="tag:example.com,2020:light",
</light/right>;rt="tag:example.com,2020:light"

Res: 2.01 Created
Location-Path: /rd/501
```

Figure 25: Example of a multicast group a CT enters into an RD

After the filling of the RD by the CT, the application in the luminaries can learn to which groups they belong, and enable their interface for the multicast address.

The luminary, knowing its sector and being configured to join any group containing lights, searches for candidate groups and joins them:

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
      ?d=R2-4-015&et=core.rd-group&rt=light

Res: 2.05 Content
Payload:
</rd/501>;ep=grp_R2-4-015;et=core.rd-group;
      base="coap://[ff05::1]";rt=core.rd-ep
```

Figure 26: Example of a lookup exchange to find suitable multicast addresses

From the returned base parameter value, the luminary learns the multicast address of the multicast group.

The presence sensor can learn the presence of groups that support resources with `rt=tag:example.com,2020:light` in its own sector by sending the same request, as used by the luminary. The presence sensor learns the multicast address to use for sending messages to the luminaries.

10.2. OMA Lightweight M2M (LwM2M)

OMA LwM2M is a profile for device services based on CoAP, providing interfaces and operations for device management and device service enablement.

An LwM2M server is an instance of an LwM2M middleware service layer, containing an RD ([LwM2M] page 36f).

That RD only implements the registration interface, and no lookup is implemented. Instead, the LwM2M server provides access to the registered resources, in a similar way to a reverse proxy.

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. Acknowledgments

Oscar Novo, Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Anders Brandt, Matthieu Vial, Jim Schaad, Mohit Sethi, Hauke Petersen, Hannes Tschofenig, Sampo Ukkola, Linyi Tian, Jan Newmarch, Matthias Kovatsch, Jaime Jimenez and Ted Lemon have provided helpful comments, discussions and ideas to improve and shape this document. Zach would also like to thank his colleagues from the EU FP7 SENSEI project, where many of the RD concepts were originally developed.

12. Changelog

changes from -27 to -28

- * Security policies / link confidentiality: Point out the RD's obligations that follow from such a policy.
- * Simple registration: clarify term "regular registration" by introducing it along with the reference to Section 5
- * Wording fix in first-come-first-remembered
- * Wording fixes in RD definition
- * Capitalization: Consistently using "registration resource"

changes from -26 to -27

- * In general, this addresses the points that were pointed out in <https://mailarchive.ietf.org/arch/msg/core/xWLomwwhovkU-CPGNxnvs40BhaM/> as having "evolved from the review comments being discussed in the interim meetings", and the review comments from Esko Dijk that were largely entangled in these points.
- * Relaxation of the serialization rules for link-format

The interpretation of RFC6690 used in Appendix B.4 was shown to be faulty. Along with a correction, the common implementations of link-format were surveyed again and it was found that the only one

that employed the faulty interpretation can still safely be upgraded. These were removed from the set considered for Limited Link Format, making the set of valid Limited Link Format documents larger.

As a consequence, the prescribed serialization of RD output can be roughly halved in bytes.

There might be additional usage patterns that are possible with the new set of constraints, but there is insufficient implementation and deployment experience with them to warrant a change changes on that front at this point. The specification can later be extended compatibly to allow these cases and drop the requirement of Limited Link Format.

- * Add Request freshness subsection

It is now recommended (with security considerations on consequences of not doing it) to require ordering of RD operations.

The Echo mechanism (previously suggested in various places but never exclusively) is the one prescribed way of getting this ordering, making the echo-request-tag reference normative.

- * Improved expression about when an RD needs to verify simple registration.

The simple wording missed the authorization part, and did not emphasize that this is a per-deployment property.

- * Point out the non-atomic properties of paginated access.

- * Clarification around impl-info reference.

- * Inconsistencies and extraneous quotations removed from examples.

changes from -25 to -26

- * Security policies:

- The First-Come-First-Remembered policy is added as an example and a potential default behavior.
- Clarify that the mapping between endpoint names and subject fields is up to a policy that defines reliance on names, and give an example.

- Random EP names: Point that multiple collisions are possible but unlikely.
- Add pointers to policies:
 - o RD replication: Point out that policies may limit that.
 - o Registration: Reword (ep, d) mapping to a previous registration's resource that could have been read as another endpoint taking over an existing registration.
- Clarify that the security policy is a property of the RD the any client may need to verify by checking the RD's authorization.
- Clarify how information from an untrusted RD can be verified
- Remove speculation about how in detail ACE scopes are obtained.
- * Security considerations:
 - Generalize to all current options for security layers usable with CoAP (OSCORE was missing as the text predated RFC8613)
 - Relax the previous SHOULD on secure access to SHOULD where protection is indicated by security policies (bringing the text in line with the -25 changes)
 - Point out that failure to follow the security considerations has implications depending on the protection objective described with the security policies
 - Shorten amplification mitigation
 - Add note about information in Registration Resource path.
 - Acknowledge that most host discovery operations are not secured; mention consequences and mitigation.
- * Abstract, introduction: removed "or disperse networks"
- * RD discovery:
 - Drop the previously stated assumption that RDAO and any DHCP options would only be used together with SLAAC and DHCP for address configuration, respectively.

- Give concrete guidance for address selection based on RFC6724 when responding to multicasts
- RDAO:
 - o Clarify that it is an option for RAs and not other ND messages.
 - o Change Lifetime from 16-bit minutes to 32-bit seconds and swap it with Reserved (aligning it with RDNSS which it shares other properties as well).
- Point out that clients may need to check RD authorization already in last discovery step
- * Registration:
 - Wording around "mostly mandatory" has been improved, conflicts clarified and sector default selection adjusted.
- * Simple registration: Rather than coopting POSTs to /.well-known/core, a new resource /.well-known/rd is registered. A historical note in the text documents the change.
- * Examples:
 - Use example URIs rather than unclear reg names (unless it's RFC6690 examples, which were kept for continuity)
 - The LwM2M example was reduced from an outdated explanation of the complete LwM2M model to a summary of how RD is used in there, with a reference to the current specification.
 - Luminary example: Explain example addresses
 - Luminary example: Drop reference to coap-group mechanism that's becoming obsolete, and thus also to RFC7390
 - Multicast addresses in the examples were changed from ff35:30:2001:db8::x to ff35:30:2001:db8:f1::8000:x; the 8000 is to follow RFC 3307, and the f1 is for consistency with all the other example addresses where 2001:db8::/32 is subnetted to 2001:db8:x::/48 by groups of internally consistent examples.
- * Use case text enhancements
 - Home and building automation: Tie in with RD

- M2M: Move system design paragraph towards the topic of reusability.
- * Various editorial fixes in response to Gen-ART and IESG reviews.
- * Rename 'Full description of the "Endpoint Type" Registration Parameter' section to '... RD Parameter'
- * Error handling: Place a SHOULD around the likely cases, and make the previous "MUST to the best of their capabilities" a "must".
- * impl-info: Add note about the type being WIP
- * Interaction tables: list CTs as possible initiators where applicable
- * Registration update: Relax requirement to not send parameters needlessly
- * Terminology: Clarify that the CTs' installation events can occur multiple times.
- * Promote RFCs 7252, 7230 and 8288 to normative references
- * Moved Christian Amsuess to first author

changes from -24 to -25

- * Large rework of section 7 (Security policies)

Rather than prescribing which data in the RD is authenticated (and how), it now describes what applications built on an RD can choose to authenticate, show possibilities on how to do it and outline what it means for clients.

This addresses Russ' Genart review points on details in the text in a rather broad fashion. That is because the discussion on the topic inside the WG showed that that text on security has been driven more review-by-review than by an architectural plan of the authors and WG.

- * Add concrete suggestions (twice as long as registrant number with retries, or UUIDs without) for random endpoint names
- * Point out that simple registration can have faked origins, RECOMMEND mitigation when applicable and suggest the Echo mechanism to implement it.

- * Reference existing and upcoming specifications for DDOS mitigation in CoAP.
 - * Explain the provenance of the example's multicast address.
 - * Make "SHOULD" of not manipulating foreign registrations a "should" and explain how it is enforced
 - * Clarify application of RFC6570 to search parameters
 - * Syntactic fixes in examples
 - * IANA:
 - Don't announce expected number of registrations (goes to write-up)
 - Include syntax as part of a field's validity in entry requirements
 - * Editorial changes
 - Align wording between abstract and introduction
 - Abbreviation normalization: "ER model", "RD"
 - RFC8174 boilerplate update
 - Minor clarity fixes
 - Markup and layouting
- changes from -23 to -24
- * Discovery using DNS-SD added again
 - * Minimum lifetime (lt) reduced from 60 to 1
 - * References added
 - * IANA considerations
 - added about .well-known/core resource
 - added DNS-SD service names
 - made RDAO option number a suggestion

- added "reference" field to endpoint type registry
 - * Lookup: mention that anchor is a legitimate lookup attribute
 - * Terminology and example fixes
 - * Layout fixes, esp. the use of non-ASCII characters in figures
- changes from -22 to -23
- * Explain that updates can not remove attributes
 - * Typo fixes
- changes from -21 to -22
- * Request a dedicated IPv4 address from IANA (rather than sharing with All CoAP nodes)
 - * Fix erroneous examples
 - * Editorial changes
- Add figure numbers to examples
 - Update RD parameters table to reflect changes of earlier versions in the text
 - Typos and minor wording
- changes from -20 to -21
- (Processing comments during WGLC)
- * Defer outdated description of using DNS-SD to find an RD to the defining document
 - * Describe operational conditions in automation example
 - * Recommend particular discovery mechanisms for some managed network scenarios
- changes from -19 to -20
- (Processing comments from the WG chair review)
- * Define the permissible characters in endpoint and sector names

- * Express requirements on NAT situations in more abstract terms
- * Shifted heading levels to have the interfaces on the same level
- * Group instructions for error handling into general section
- * Simple Registration: process reflowed into items list
- * Updated introduction to reflect state of CoRE in general, reference RFC7228 (defining "constrained") and use "IoT" term in addition to "M2M"
- * Update acknowledgements
- * Assorted editorial changes
 - Unify examples style
 - Terminology: RDAO defined and not only expanded
 - Add CT to Figure 1
 - Consistency in the use of the term "Content Format"

changes from -18 to -19

- * link-local addresses: allow but prescribe split-horizon fashion when used, disallow zone identifiers
- * Remove informative references to documents not mentioned any more

changes from -17 to -18

- * Rather than re-specifying link format (Modernized Link Format), describe a Limited Link Format that's the uncontested subset of Link Format
- * Acknowledging the -17 version as part of the draft
- * Move "Read endpoint links" operation to future specification like PATCH
- * Demote links-json to an informative reference, and removed them from exchange examples
- * Add note on unusability of link-local IP addresses, and describe mitigation.

- * Reshuffling of sections: Move additional operations and endpoint lookup back from appendix, and groups into one
- * Lookup interface tightened to not imply applicability for non link-format lookups (as those can have vastly different views on link cardinality)
- * Simple registration: Change sequence of GET and POST-response, ensuring unsuccessful registrations are reported as such, and suggest how devices that would have required the inverse behavior can still cope with it.
- * Abstract and introduction reworded to avoid the impression that resources are stored in full in the RD
- * Simplify the rules governing when a registration resource can or must be changed.
- * Drop a figure that has become useless due to the changes of and -13 and -17
- * Wording consistency fixes: Use "Registrations" and "target attributes"
- * Fix incorrect use of content negotiation in discovery interface description (Content-Format -> Accept)
- * State that the base attribute value is part of endpoint lookup even when implicit in the registration
- * Update references from RFC5988 to its update RFC8288
- * Remove appendix on protocol-negotiation (which had a note to be removed before publication)

changes from -16 to -17

(Note that -17 is published as a direct follow-up to -16, containing a single change to be discussed at IETF103)

- * Removed groups that are enumerations of registrations and have dedicated mechanism
- * Add groups that are enumerations of shared resources and are a special case of endpoint registrations

changes from -15 to -16

- * Recommend a common set of resources for members of a group
- * Clarified use of multicast group in lighting example
- * Add note on concurrent registrations from one EP being possible but not expected
- * Refresh web examples appendix to reflect current use of Modernized Link Format
- * Add examples of URIs where Modernized Link Format matters
- * Editorial changes

changes from -14 to -15

- * Rewrite of section "Security policies"
- * Clarify that the "base" parameter text applies both to relative references both in anchor and href
- * Renamed "Registree-EP" to Registrant-EP"
- * Talk of "relative references" and "URIs" rather than "relative" and "absolute" URIs. (The concept of "absolute URIs" of [RFC3986] is not needed in RD).
- * Fixed examples
- * Editorial changes

changes from -13 to -14

- * Rename "registration context" to "registration base URI" (and "con" to "base") and "domain" to "sector" (where the abbreviation "d" stays for compatibility reasons)
- * Introduced resource types core.rd-ep and core.rd-gp
- * Registration management moved to appendix A, including endpoint and group lookup
- * 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
- * Simple registration carries no error information and succeeds immediately (previously, sequence was unspecified)
- * Lookup: href are matched against resolved values (previously, this was unspecified)
- * Lookup: lt are not exposed any more
- * con/base: Paths are allowed
- * Registration resource locations can not have query or fragment parts
- * Default life time extended to 25 hours
- * clarified registration update rules
- * lt-value semantics for lookup clarified.
- * added template for simple registration

changes from -12 to -13

- * Added "all resource directory" nodes MC address
- * Clarified observation behavior
- * version identification
- * example rt= and et= values
- * domain from figure 2
- * more explanatory text
- * endpoints of a groups hosted by different RD
- * 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

- * added Content Model section, including ER diagram
- * removed domain lookup interface; domains are now plain attributes of groups and endpoints
- * updated chapter "Finding a Resource Directory"; now distinguishes configuration-provided, network-provided and heuristic sources
- * improved text on: atomicity, idempotency, lookup with multiple parameters, endpoint removal, simple registration
- * updated LWM2M description
- * clarified where relative references are resolved, and how context and anchor interact
- * new appendix on the interaction with RFCs 6690, 5988 and 3986
- * lookup interface: group and endpoint lookup return group and registration resources as link targets
- * lookup interface: search parameters work the same across all entities
- * removed all methods that modify links in an existing registration (POST with payload, PATCH and iPATCH)
- * removed plurality definition (was only needed for link modification)
- * enhanced IANA registry text
- * state that lookup resources can be observable
- * More examples and improved text

changes from -09 to -10

- * removed "ins" and "exp" link-format extensions.
- * removed all text concerning DNS-SD.
- * removed inconsistency in RDAO text.
- * suggestions taken over from various sources
- * replaced "Function Set" with "REST API", "base URI", "base path"

- * moved simple registration to registration section

changes from -08 to -09

- * clarified the "example use" of the base RD resource values /rd, /rd-lookup, and /rd-group.
- * changed "ins" ABNF notation.
- * various editorial improvements, including in examples
- * clarifications for RDAO

changes from -07 to -08

- * removed link target value returned from domain and group lookup types
- * Maximum length of domain parameter 63 bytes for consistency with group
- * 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
- * add IPv6 ND Option for discovery of an RD
- * clarify group configuration section 6.1 that endpoints must be registered before including them in a group
- * removed all superfluous client-server diagrams
- * simplified lighting example
- * introduced Commissioning Tool
- * RD-Look-up text is extended.

changes from -06 to -07

- * added text in the discovery section to allow content format hints to be exposed in the discovery link attributes
- * editorial updates to section 9
- * update author information
- * minor text corrections

Changes from -05 to -06

- * added note that the PATCH section is contingent on the progress of the PATCH method

changes from -04 to -05

- * added Update Endpoint Links using PATCH
- * http access made explicit in interface specification
- * Added http examples

Changes from -03 to -04:

- * Added http response codes
- * Clarified endpoint name usage
- * Add application/link-format+cbor content-format

Changes from -02 to -03:

- * Added an example for lighting and DNS integration
- * Added an example for RD use in OMA LWM2M
- * Added Read Links operation for link inspection by endpoints
- * Expanded DNS-SD section
- * Added draft authors Peter van der Stok and Michael Koster

Changes from -01 to -02:

- * Added a catalogue use case.
- * Changed the registration update to a POST with optional link format payload. Removed the endpoint type update from the update.
- * Additional examples section added for more complex use cases.
- * New DNS-SD mapping section.
- * Added text on endpoint identification and authentication.
- * Error code 4.04 added to Registration Update and Delete requests.

- * Made 63 bytes a SHOULD rather than a MUST for endpoint name and resource type parameters.

Changes from -00 to -01:

- * Removed the ETag validation feature.
- * Place holder for the DNS-SD mapping section.
- * Explicitly disabled GET or POST on returned Location.
- * New registry for RD parameters.
- * Added support for the JSON Link Format.
- * Added reference to the Groupcomm WG draft.

Changes from -05 to WG Document -00:

- * Updated the version and date.

Changes from -04 to -05:

- * Restricted Update to parameter updates.
- * Added pagination support for the Lookup interface.
- * Minor editing, bug fixes and reference updates.
- * Added group support.
- * Changed rt to et for the registration and update interface.

Changes from -03 to -04:

- * Added the ins= parameter back for the DNS-SD mapping.
- * Integrated the Simple Directory Discovery from Carsten.
- * Editorial improvements.
- * Fixed the use of ETags.
- * Fixed tickets 383 and 372

Changes from -02 to -03:

- * Changed the endpoint name back to a single registration parameter ep= and removed the h= and ins= parameters.
- * Updated REST interface descriptions to use RFC6570 URI Template format.
- * Introduced an improved RD Lookup design as its own function set.
- * Improved the security considerations section.
- * Made the POST registration interface idempotent by requiring the ep= parameter to be present.

Changes from -01 to -02:

- * Added a terminology section.
- * Changed the inclusion of an ETag in registration or update to a MAY.
- * Added the concept of an RD Domain and a registration parameter for it.
- * Recommended the Location returned from a registration to be stable, allowing for endpoint and Domain information to be changed during updates.
- * Changed the lookup interface to accept endpoint and Domain as query string parameters to control the scope of a lookup.

13. References

13.1. Normative References

- [I-D.ietf-core-echo-request-tag]
Amsüss, C., Mattsson, J. P., and G. Selander, "CoAP: Echo, Request-Tag, and Token Processing", Work in Progress, Internet-Draft, draft-ietf-core-echo-request-tag-12, 1 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-echo-request-tag-12.txt>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.

- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <<https://www.rfc-editor.org/info/rfc6570>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<https://www.rfc-editor.org/info/rfc6690>>.
- [RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <<https://www.rfc-editor.org/info/rfc6763>>.
- [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>>.
- [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>>.
- [RFC8288] Nottingham, M., "Web Linking", RFC 8288, DOI 10.17487/RFC8288, October 2017, <<https://www.rfc-editor.org/info/rfc8288>>.

13.2. Informative References

- [ER] Chen, P., "The entity-relationship model--toward a unified view of data", DOI 10.1145/320434.320440, ACM Transactions on Database Systems Vol. 1, pp. 9-36, March 1976, <<https://doi.org/10.1145/320434.320440>>.

[I-D.bormann-t2trg-rel-impl]

Bormann, C., "impl-info: A link relation type for disclosing implementation information", Work in Progress, Internet-Draft, draft-bormann-t2trg-rel-impl-02, 27 September 2020, <<https://www.ietf.org/archive/id/draft-bormann-t2trg-rel-impl-02.txt>>.

[I-D.hartke-t2trg-coral]

Hartke, K., "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-hartke-t2trg-coral-09, 8 July 2019, <<https://www.ietf.org/archive/id/draft-hartke-t2trg-coral-09.txt>>.

[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)", Work in Progress, Internet-Draft, draft-ietf-ace-oauth-authz-37, 4 February 2021, <<https://www.ietf.org/archive/id/draft-ietf-ace-oauth-authz-37.txt>>.

[I-D.ietf-core-links-json]

LI, K., Rahman, A., and C. Bormann, "Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR", Work in Progress, Internet-Draft, draft-ietf-core-links-json-10, 26 February 2018, <<https://www.ietf.org/archive/id/draft-ietf-core-links-json-10.txt>>.

[I-D.ietf-core-rd-dns-sd]

Stok, P. V. D., Koster, M., and C. Amsüss, "CoRE Resource Directory: DNS-SD mapping", Work in Progress, Internet-Draft, draft-ietf-core-rd-dns-sd-05, 7 July 2019, <<https://www.ietf.org/archive/id/draft-ietf-core-rd-dns-sd-05.txt>>.

[I-D.silverajan-core-coap-protocol-negotiation]

Silverajan, B. and M. Ocak, "CoAP Protocol Negotiation", Work in Progress, Internet-Draft, draft-silverajan-core-coap-protocol-negotiation-09, 2 July 2018, <<https://www.ietf.org/archive/id/draft-silverajan-core-coap-protocol-negotiation-09.txt>>.

[LwM2M]

Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification: Transport Bindings (Candidate Version 1.1)", 12 June 2018,

- https://openmobilealliance.org/RELEASE/LightweightM2M/V1_1-20180612-C/OMA-TS-LightweightM2M_Transport-V1_1-20180612-C.pdf.
- [RFC3306] Haberman, B. and D. Thaler, "Unicast-Prefix-based IPv6 Multicast Addresses", RFC 3306, DOI 10.17487/RFC3306, August 2002, <https://www.rfc-editor.org/info/rfc3306>.
- [RFC3849] Huston, G., Lord, A., and P. Smith, "IPv6 Address Prefix Reserved for Documentation", RFC 3849, DOI 10.17487/RFC3849, July 2004, <https://www.rfc-editor.org/info/rfc3849>.
- [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>.
- [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>.
- [RFC5771] Cotton, M., Vegoda, L., and D. Meyer, "IANA Guidelines for IPv4 Multicast Address Assignments", BCP 51, RFC 5771, DOI 10.17487/RFC5771, March 2010, <https://www.rfc-editor.org/info/rfc5771>.
- [RFC6724] Thaler, D., Ed., Draves, R., Matsumoto, A., and T. Chown, "Default Address Selection for Internet Protocol Version 6 (IPv6)", RFC 6724, DOI 10.17487/RFC6724, September 2012, <https://www.rfc-editor.org/info/rfc6724>.
- [RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <https://www.rfc-editor.org/info/rfc6775>.
- [RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <https://www.rfc-editor.org/info/rfc6874>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

- [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>>.
- [RFC8106] Jeong, J., Park, S., Beloeil, L., and S. Madanapalli, "IPv6 Router Advertisement Options for DNS Configuration", RFC 8106, DOI 10.17487/RFC8106, March 2017, <<https://www.rfc-editor.org/info/rfc8106>>.
- [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>>.
- [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>>.
- [RFC8613] Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", RFC 8613, DOI 10.17487/RFC8613, July 2019, <<https://www.rfc-editor.org/info/rfc8613>>.

Appendix A. Groups Registration and Lookup

The RD-Groups usage pattern allows announcing application groups inside an RD.

Groups are represented by endpoint registrations. Their base address is a multicast address, and they SHOULD be entered with the endpoint type "core.rd-group". The endpoint name can also be referred to as a group name in this context.

The registration is inserted into the RD by a Commissioning Tool, which might also be known as a group manager here. It performs third party registration and registration updates.

The links it registers SHOULD be available on all members that join the group. Depending on the application, members that lack some resource MAY be permissible if requests to them fail gracefully.

The following example shows a CT registering a group with the name "lights" which provides two resources. The directory resource path /rd is an example RD location discovered in a request similar to Figure 5. The group address in the example is constructed from [RFC3849]'s reserved 2001:db8:: prefix as a unicast-prefix based site-local address (see [RFC3306]).

```

Req: POST coap://rd.example.com/rd?ep=lights&et=core.rd-group
      &base=coap://[ff35:30:2001:db8:f1::8000:1]
Content-Format: 40
Payload:
</light>;rt="tag:example.com,2020:light";
      if="tag:example.net,2020:actuator",
</color-temperature>;if="tag:example.net,2020:parameter";u=K

Res: 2.01 Created
Location-Path: /rd/12

```

Figure 27: Example registration of a group

In this example, the group manager can easily permit devices that have no writable color-temperature to join, as they would still respond to brightness changing commands. Had the group instead contained a single resource that sets brightness and color temperature atomically, endpoints would need to support both properties.

The resources of a group can be looked up like any other resource, and the group registrations (along with any additional registration parameters) can be looked up using the endpoint lookup interface.

The following example shows a client performing an endpoint lookup for all groups.

```

Req: GET /rd-lookup/ep?et=core.rd-group

Res: 2.05 Content
Payload:
</rd/12>;ep=lights&et=core.rd-group;
      base="coap://[ff35:30:2001:f1:db8::8000:1]";rt=core.rd-ep

```

Figure 28: Example lookup of groups

The following example shows a client performing a lookup of all resources of all endpoints (groups) with et=core.rd-group.

```

Req: GET /rd-lookup/res?et=core.rd-group

Res: 2.05 Content
Payload:
<coap://[ff35:30:2001:db8:f1::8000:1]/light>;
      rt="tag:example.com,2020:light";
      if="tag:example.net,2020:actuator",
<coap://[ff35:30:2001:db8:f1::8000:1]/color-temperature>;
      if="tag:example.net,2020:parameter";u=K,

```

Figure 29: Example lookup of resources inside groups

Appendix B. Web links and the Resource Directory

Understanding the semantics of a link-format document and its URI references is a journey through different documents ([RFC3986] defining URIs, [RFC6690] defining link-format documents based on [RFC8288] which defines Link header fields, 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:

```
Req: GET coap://[ff02::fd]:5683/.well-known/core?rt=temperature
```

```
Res: 2.05 Content
```

```
Payload:
```

```
</sensors/temp>;rt=temperature;ct=0
```

Figure 30: Example of direct resource discovery

where the response is sent by the server, `"[2001:db8:f0::1]:5683"`.

While the client -- on the practical or implementation side -- can just go ahead and create a new request to `"[2001:db8:f0::1]:5683"` with Uri-Path: `"sensors"` and `"temp"`, the full resolution steps for insertion into and retrieval from the RD without any shortcuts are:

B.1.1. Resolving the URIs

The client parses the single returned record. The link's target (sometimes called `"href"`) is `"/sensors/temp"`, which is a relative URI that needs resolving. The base URI `<coap://[ff02::fd]:5683/.well-known/core>` is used to resolve the reference `/sensors/temp` against.

The Base URI of the requested resource can be composed from the options of the CoAP GET request by following the steps of [RFC7252] section 6.5 (with an addition at the end of 8.2) into `"coap://[2001:db8:f0::1]/.well-known/core"`.

Because `"/sensors/temp"` starts with a single slash, the record's target is resolved by replacing the path `"/.well-known/core"` from the Base URI (section 5.2 [RFC3986]) with the relative target URI `"/sensors/temp"` into `"coap://[2001:db8:f0::1]/sensors/temp"`.

B.1.2. Interpreting attributes and relations

Some more information but the record's target can be obtained from the payload: the resource type of the target is "temperature", and its content format is text/plain (ct=0).

A relation in a web link is a three-part statement that specifies a named relation between the so-called "context resource" and the target resource, like `"_This page_ has _its table of contents_ at _/toc.html_"`. In link format documents, there is an implicit "host relation" specified with default parameter: `rel="hosts"`.

In our example, the context resource of the link is implied to be `"coap://[2001:db8:f0::1]"` by the default value of the anchor (see Appendix B.4). A full English expression of the "host relation" is:

`'"coap://[2001:db8:f0::1]" is hosting the resource "coap://[2001:db8:f0::1]/sensors/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:

Req: GET `coap://[ff02::fd]:5683/.well-known/core`

Res: 2.05 Content

Payload:

```
</sensors/temp>;rt=temperature;ct=0,
</sensors/light>;rt=light-lux;ct=0,
</t>;anchor="/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;anchor="/sensors/temp";
rel=describedby
```

Figure 31: Extended example of direct resource discovery

Parsing the third record, the client encounters the "anchor" parameter. It is a URI relative to the Base URI of the request and is thus resolved to `"coap://[2001:db8:f0::1]/sensors/temp"`. That is the context resource of the link, so the "rel" statement is not about the target and the Base URI any more, but about the target and the resolved URI. Thus, the third record could be read as `"coap://[2001:db8:f0::1]/sensors/temp"` has an alternate representation at `"coap://[2001:db8:f0::1]/t"`.

Following the same resolution steps, the fourth record can be read as `"coap://[2001:db8:f0::1]/sensors/temp"` is described by `"http://www.example.com/sensors/t123"`.

B.3. Enter the Resource Directory

The RD 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 RD that was announced to it, sending this request from its UDP port `"[2001:db8:f0::1]:6553"`:

Req: POST `coap://[2001:db8:f01::ff]/.well-known/rd?ep=simple-host1`

Res: 2.04 Changed

Figure 32: Example of a simple registration

The RD would have accepted the registration, and queried the simple host's `"/.well-known/core"` by itself. As a result, the host is registered as an endpoint in the RD with the name `"simple-host1"`. The registration is active for 90000 seconds, and the endpoint registration Base URI is `"coap://[2001:db8:f0::1]"` following the resolution steps described in Appendix B.1.1. It should be remarked that the Base URI constructed that way always yields a URI of the form: `scheme://authority without path suffix`.

If the client now queries the RD as it would previously have issued a multicast request, it would go through the RD discovery steps by fetching `"coap://[2001:db8:f0::ff]/.well-known/core?rt=core.rd-lookup-res"`, obtain `"coap://[2001:db8:f0::ff]/rd-lookup/res"` as the resource lookup endpoint, and ask it for all temperature resources:

```
Req: GET coap://[2001:db8:f0::ff]/rd-lookup/res?rt=temperature
```

```
Res: 2.05 Content
```

```
Payload:
```

```
<coap://[2001:db8:f0::1]/sensors/temp>;rt=temperature;ct=0
```

Figure 33: Example exchange performing resource lookup

This is not literally the same response that it would have received from a multicast request, but it contains the equivalent statement:

```
'"coap://[2001:db8:f0::1]" is hosting the resource
"coap://[2001:db8:f0::1]/sensors/temp", which is of the resource type
"temperature" and can be accessed using the text/plain content
format.'
```

To complete the examples, the client could also query all resources hosted at the endpoint with the known endpoint name "simple-host1":

```
Req: GET coap://[2001:db8:f0::ff]/rd-lookup/res?ep=simple-host1
```

```
Res: 2.05 Content
```

```
Payload:
```

```
<coap://[2001:db8:f0::1]/sensors/temp>;rt=temperature;ct=0,
<coap://[2001:db8:f0::1]/sensors/light>;rt=light-lux;ct=0,
<coap://[2001:db8:f0::1]/t>;
  anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;
  anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=describedby
```

Figure 34: Extended example exchange performing resource lookup

All the target and anchor references are already in absolute form there, which don't need to be resolved any further.

Had the simple host done an equivalent full registration with a base= parameter (e.g. "?ep=simple-host1&base=coap+tcp://simple-host1.example.com"), that context would have been used to resolve the relative anchor values instead, giving

```
<coap+tcp://simple-host1.example.com/sensors/temp>;rt=temperature;ct=0
```

Figure 35: Example payload of a response to a resource lookup with a dedicated base URI

and analogous records.

B.4. A note on differences between link-format and Link header fields

While link-format and Link header fields look very similar and are based on the same model of typed links, there are some differences between [RFC6690] and [RFC8288]. When implementing an RD or interacting with an RD, care must be taken to follow the [RFC6690] behavior whenever application/link-format representations are used.

- * "Default value of anchor": Both under [RFC6690] and [RFC8288], relative references in the term inside the angle brackets (the target) and the anchor attribute are resolved against the relevant base URI (which usually is the URI used to retrieve the entity), and independent of each other.

When, in an [RFC8288] Link header, the anchor attribute is absent, the link's context is the URI of the selected representation (and usually equal to the base URI).

In [RFC6690] links, if the anchor attribute is absent, the default value is the Origin of (for all relevant cases: the URI reference "/" resolved against) the link's target.

- * 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 header fields are practically ASCII strings that use percent encoding for non-ASCII characters, stating the encoding explicitly when required.

For example, while a Link header field 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/Malmö>;rel=live-environment-data
```

Appendix C. Limited Link Format

The CoRE Link Format as described in [RFC6690] has been interpreted differently by implementers, and a strict implementation rules out some use cases of an RD (e.g. base values with path components in combination with absent anchors).

This appendix describes a subset of link format documents called Limited Link Format. The one rule herein is not very limiting in practice -- all examples in RFC6690, and all deployments the authors are aware of already stick to them -- but ease the implementation of RD servers.

It is applicable to representations in the application/link-format media type, and any other media types that inherit [RFC6690] Section 2.1.

A link format representation is in Limited Link format if, for each link in it, the following applies:

All URI references either follow the URI or the path-absolute ABNF rule of RFC3986 (i.e. target and anchor each either start with a scheme or with a single slash).

Authors' Addresses

Christian Amsüss (editor)
Hollandstr. 12/4
1020
Austria

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

Zach Shelby
ARM
150 Rose Orchard
San Jose, 95134
United States of America

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View, 94043
United States of America

Phone: +1-707-502-5136
Email: Michael.Koster@smarththings.com

Carsten Bormann
Universitaet Bremen TZI
Postfach 330440
D-28359 Bremen
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

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	vb	4	Boolean	boolean	a	RFC-AAAA
Value						
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. 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/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: 22 May 2022

M.V. Veillette, Ed.
Trilliant Networks Inc.
A.P. Pelov, Ed.
Acklio
I. Petrov, Ed.
Google Switzerland GmbH
C. Bormann
Universität Bremen TZI
M. Richardson
Sandelman Software Works
18 November 2021

YANG Schema Item iDentifier (YANG SID)
draft-ietf-core-sid-18

Abstract

YANG Schema Item iDentifiers (YANG SID) are globally unique 63-bit unsigned integers used to identify YANG items, as a more compact method to identify YANG items that can be used for efficiency and in constrained environments (RFC 7228). This document defines the semantics, the registration, and assignment processes of YANG SIDs for IETF managed YANG modules. To enable the implementation of these processes, this document also defines a file format used to persist and publish assigned YANG 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 22 May 2022.

Copyright Notice

Copyright (c) 2021 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Terminology and Notation	4
3. ".sid" file lifecycle	5
4. ".sid" file format	6
5. Content-Types	13
6. Security Considerations	13
7. IANA Considerations	14
7.1. YANG Namespace Registration	14
7.2. Register ".sid" File Format Module	14
7.3. Create new IANA Registry: "YANG SID Mega-Range" registry	15
7.3.1. Structure	15
7.3.2. Allocation policy	15
7.3.2.1. First allocation	16
7.3.2.2. Consecutive allocations	16
7.3.3. Initial contents of the Registry	16
7.4. Create a new IANA Registry: IETF YANG SID Range Registry (managed by IANA)	17
7.4.1. Structure	17
7.4.2. Allocation policy	17
7.4.3. Publication of the ".sid" file	18
7.4.4. Initial contents of the registry	19
7.5. Create new IANA Registry: "IETF YANG SID Registry" . . .	21
7.5.1. Structure	21
7.5.2. Allocation policy	21
7.5.3. Recursive Allocation of YANG SID Range at Document Adoption	22
7.5.4. Initial contents of the registry	23
8. References	23
8.1. Normative References	23
8.2. Informative References	24
Appendix A. ".sid" file example	26
Appendix B. SID auto generation	36
Appendix C. ".sid" file lifecycle	37
C.1. ".sid" File Creation	37
C.2. ".sid" File Update	38
Acknowledgments	39

Contributors	40
Authors' Addresses	40

1. Introduction

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both Network Configuration Protocol (NETCONF) [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using names. To allow the implementation of data models defined in YANG in constrained devices [RFC7228] and constrained networks, a more compact method to identify YANG items is required. This compact identifier, called YANG Schema Item iDentifier or YANG SID (or simply SID in this document and when the context is clear), is encoded using a 63-bit unsigned integer. The limitation to 63-bit unsigned integers allows SIDs to be manipulated more easily on platforms that might otherwise lack 64-bit unsigned arithmetic. The loss of a single bit of range is not significant given the size of the remaining space.

The following items are identified using SIDs:

- * identities
- * data nodes (Note: including those nodes defined by the 'yang-data' extension.)
- * remote procedure calls (RPCs) and associated input(s) and output(s)
- * actions and associated input(s) and output(s)
- * notifications and associated information
- * YANG modules and features

It is possible that some protocols use only a subset of the assigned SIDs, for example, for protocols equivalent to NETCONF [RFC6241] like [I-D.ietf-core-comi] the transportation of YANG module SIDs might be unnecessary. Other protocols might need to be able to transport this information, for example protocols related to discovery such as Constrained YANG Module Library [I-D.ietf-core-yang-library].

SIDs are globally unique integers. A registration system is used in order to guarantee their uniqueness. SIDs are registered in blocks called "SID ranges".

SIDs are assigned permanently. Items introduced by a new revision of a YANG module are added to the list of SIDs already assigned. Assignment of SIDs to YANG items are usually automated as discussed in Appendix B, which also discusses some cases where manual interventions may be appropriate.

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.

IETF managed YANG modules that need to allocate SIDs use the IANA mechanism specified in this document. YANG modules created by other parties allocate SID ranges using the IANA allocation mechanisms via Mega-Ranges (see Section 7.3); within the Mega-Range allocation, those other parties are free to make up their own mechanism.

At the time of writing, a tool for automated ".sid" file generation is available as part of the open-source project PYANG [PYANG].

2. Terminology and Notation

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

The following terms are defined in [RFC7950]:

- * action
- * feature
- * module
- * notification
- * RPC
- * schema node
- * schema tree
- * submodule

The following term is defined in [RFC8040]:

- * yang-data extension

This specification also makes use of the following terminology:

- * item: A schema node, an identity, a module, or a feature defined using the YANG modeling language.
- * schema-node path: A schema-node path is a string that identifies a schema node within the schema tree. A path consists of the list of consecutive 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 and could contain namespace information. (e.g. "/ietf-system:system-state/clock/current-datetime")
- * Namespace-qualified form - a schema node identifier is prefixed with the name of the module in which the schema node is defined, separated from the schema node identifier by the colon character (":").
- * YANG Schema Item iDentifier (YANG SID or simply 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], RESTCONF [RFC8040] and CORECONF [I-D.ietf-core-comi]). A YANG module defines hierarchies of data, including configuration, state data, RPCs, actions and notifications.

Many YANG modules are not created in the context of constrained applications. YANG modules can be implemented using NETCONF [RFC6241] or RESTCONF [RFC8040] without the need to assign SIDs.

As needed, authors of YANG modules can assign SIDs to their YANG modules. In order to do that, they should first obtain a SID range from a registry and use that range to assign or generate SIDs to items of their YANG module. The assignments can then be stored in a ".sid" file. For example on how this could be achieved, please refer to Appendix C.

Registration of the ".sid" file associated to a YANG module is optional but recommended to promote interoperability between devices and to avoid duplicate allocation of SIDs to a single YANG module. Different registries might have different requirements for the registration and publication of the ".sid" files. For a diagram of one of the possibilities, please refer to the activity diagram on Figure 4 in Appendix C.

Each time a YANG module or one of its imported module(s) or included sub-module(s) is updated, a new ".sid" file MAY be created if the new or updated items will need SIDs. All the SIDs present in the previous version of the ".sid" file MUST be present in the new version as well. The creation of this new version of the ".sid" file SHOULD 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-range' as defined in Section 4. These extra SIDs are used for subsequent assignments.

For an example of this update process, see activity diagram Figure 5 in Appendix C.

4. ".sid" file format

".sid" files are used to persist and publish SIDs assigned to the different YANG items of a specific YANG module. It has the following structure.

module: ietf-sid-file

```

structure sid-file:
  +-- module-name          yang:yang-identifier
  +-- module-revision?     revision-identifier
  +-- sid-file-version?    sid-file-version-identifier
  +-- description?         string
  +-- dependency-revision* [module-name]
    |   +-- module-name      yang:yang-identifier
    |   +-- module-revision  revision-identifier
  +-- assignment-range* [entry-point]
    |   +-- entry-point      sid
    |   +-- size             uint64
  +-- item* [namespace identifier]
    +-- namespace          enumeration
    +-- identifier          union
    +-- sid                 sid

```

Figure 1: YANG tree for ietf-sid-file

The following YANG module defines the structure of this file, encoding is performed in JSON [RFC8259] using the rules defined in [RFC7951]. It references ietf-yang-types defined in [RFC6991] and ietf-restconf defined in [RFC8040].

RFC Ed.: please update the date of the module and Copyright if needed and remove this note.

```
<CODE BEGINS> file "ietf-sid-file@2021-11-16.yang"
module ietf-sid-file {
  yang-version 1.1;
  namespace "urn:ietf:params:xml:ns:yang:ietf-sid-file";
  prefix sid;

  import ietf-yang-types {
    prefix yang;
    reference "RFC 6991: Common YANG Data Types.";
  }
  import ietf-yang-structure-ext {
    prefix sx;
    reference "RFC 8791: YANG Data Structure Extensions.";
  }

  organization
    "IETF Core Working Group";

  contact
    "WG Web:  <https://datatracker.ietf.org/wg/core/>

    WG List:  <mailto:core@ietf.org>

    Editor:    Michel Veillette
               <mailto:michel.veillette@trilliant.com>

    Editor:    Andy Bierman
               <mailto:andy@yumaworks.com>

    Editor:    Alexander Pelov
               <mailto:a@ackl.io>

    Editor:    Ivaylo Petrov
               <mailto:ivaylopetrov@google.com>";

  description
    "Copyright (c) 2021 IETF Trust and the persons identified as
    authors of the code.  All rights reserved.

    Redistribution and use in source and binary forms, with or
```

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

This version of this YANG module is part of RFC XXXX (<https://www.rfc-editor.org/info/rfcXXXX>); see the RFC itself for full legal notices.

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 (RFC 2119) (RFC 8174) when, and only when, they appear in all capitals, as shown here.

This module defines the structure of the .sid files.

Each .sid file contains the mapping between each string identifier defined by a YANG module and a corresponding numeric value called YANG SID.";

```
revision 2021-11-16 {
  description
    "Initial revision.";
  reference
    "[RFC XXXX] YANG Schema Item iDentifier (YANG SID)";
}

typedef revision-identifier {
  type string {
    pattern '\d{4}-\d{2}-\d{2}';
  }
  description
    "Represents a date in YYYY-MM-DD format.";
}

typedef sid-file-version-identifier {
  type uint32;
  description
    "Represents the version of a .sid file.";
}

typedef sid {
  type uint64 {
    range "0..9223372036854775807";
  }
  description
```

```

    "YANG Schema Item iDentifier";
  reference
    "[RFC XXXX] YANG Schema Item iDentifier (YANG SID)";
}

typedef schema-node-path {
  type string {
    pattern
      '/[a-zA-Z_][a-zA-Z0-9\-\_\.]*:[a-zA-Z_][a-zA-Z0-9\-\_\.]*' +
      '(/[a-zA-Z_][a-zA-Z0-9\-\_\.]*(:[a-zA-Z_][a-zA-Z0-9\-\_\.]*)?)?)*';
  }
  description
    "A schema-node path is an absolute YANG schema node identifier
    as defined by the YANG ABNF rule absolute-schema-nodeid,
    except that module names are used instead of prefixes.

    This string additionally follows the following rules:

    o The leftmost (top-level) data node name is always in the
      namespace-qualified form.
    o Any subsequent schema node name is in the
      namespace-qualified form if the node is defined in a module
      other than its parent node, and the simple form is used
      otherwise. No predicates are allowed.";
  reference
    "RFC 7950, The YANG 1.1 Data Modeling Language;
    Section 6.5: Schema Node Identifier";
}

sx:structure sid-file {
  uses sid-file-contents;
}

grouping sid-file {
  description "A grouping that contains a YANG container
  representing the file structure of a .sid files.";

  container sid-file {
    description
      "A Wrapper container that together with the rc:yang-data
      extension marks the YANG data structures inside as not being
      intended to be implemented as part of a configuration
      datastore or as an operational state within the server.";
    uses sid-file-contents;
  }
}

grouping sid-file-contents {

```



```
description
  "A grouping that defines the contents of a container that
  represente the file structure of a .sid files.";

leaf module-name {
  type yang:yang-identifier;
  mandatory true;
  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.";
}

leaf sid-file-version {
  type sid-file-version-identifier;
  default 0;
  description
    "Optional leaf that specifies the version number of the
    .sid file. .sid files and the version sequence are
    specific to a given YANG module revision. This number
    starts at zero when there is a new YANG module revision and
    increases monotonically. This number can distinguish
    updates to the .sid file which are the result of new
    processing, or the result of reported errata.";
}

leaf description {
  type string;
  description
    "Free-form meta information about the generated file. It
    might include .sid file generation tool and time among
    other things.";
}

list dependency-revision {
  key "module-name";

  description
    "Information about the used revision during the .sid file
    generation of each YANG module that the module in
    'module-name' imported.";
```

```

    leaf module-name {
        type yang:yang-identifier;
        description
            "Name of the YANG module, dependency of 'module-name',
            for which revision information is provided.";
    }
    leaf module-revision {
        type revision-identifier;
        mandatory true;
        description
            "Revision of the YANG module, dependency of
            'module-name', for which revision information is
            provided.";
    }
}

list assignment-range {
    key "entry-point";
    description
        "YANG SID range(s) allocated to the YANG module identified
        by 'module-name' and 'module-revision'."

        - The YANG SID range first available value is entry-point
          and the last available value in the range is
          (entry-point + size - 1).
        - The YANG SID ranges specified by all assignment-rages
          MUST NOT overlap.";

    leaf entry-point {
        type sid;
        description
            "Lowest YANG SID available for assignment.";
    }

    leaf size {
        type uint64;
        mandatory true;
        description
            "Number of YANG SIDs available for assignment.";
    }
}

list item {
    key "namespace identifier";
    unique "sid";

    description
        "Each entry within this list defined the mapping between

```

a YANG item string identifier and a YANG SID. This list MUST include a mapping entry for each YANG item defined by the YANG module identified by 'module-name' and 'module-revision'.";

```
leaf namespace {
  type enumeration {
    enum module {
      value 0;
      description
        "All module and submodule names share the same
        global module identifier namespace.";
    }
    enum identity {
      value 1;
      description
        "All identity names defined in a module and its
        submodules share the same identity identifier
        namespace.";
    }
    enum feature {
      value 2;
      description
        "All feature names defined in a module and its
        submodules share the same feature identifier
        namespace.";
    }
    enum data {
      value 3;
      description
        "The namespace for all data nodes, as defined in
        YANG.";
    }
  }
  description
    "Namespace of the YANG item for this mapping entry.";
}

leaf identifier {
  type union {
    type yang:yang-identifier;
    type schema-node-path;
  }
  description
    "String identifier of the YANG item for this mapping
    entry.

    If the corresponding 'namespace' field is 'module',
```

```

        'feature', or 'identity', then this field MUST
        contain a valid YANG identifier string.

        If the corresponding 'namespace' field is 'data',
        then this field MUST contain a valid schema node
        path.";
    }

    leaf sid {
        type sid;
        mandatory true;
        description
            "YANG SID assigned to the YANG item for this mapping
            entry.";
    }
}
}
}
<CODE ENDS>

```

Figure 2: YANG module ietf-sid-file

5. Content-Types

The following Content-Type has been defined in [I-D.ietf-core-yang-cbor]:

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

FORMAT: CBOR map of SID, instance-value

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

6. Security Considerations

This document defines a new type of identifier used to encode data that are modeled in YANG [RFC7950]. This new identifier maps semantic concepts to integers, and if the source of this mapping is not trusted, then new security risks might occur if an attacker can control the mapping.

At the time of writing, it is expected that the SID files will be processed by a software developer, within a software development environment. Developers are advised to only import SID files from authoritative sources. IANA is the authoritative source for IETF managed YANG modules.

Conceptually, SID files could be processed by less-constrained target systems such as network management systems. Such systems need to take extra care to make sure that they are only processing SID files from authoritative sources, as authoritative as the YANG modules that they are using.

7. IANA Considerations

7.1. YANG Namespace Registration

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

URI: please assign urn:ietf:params:xml:ns:yang:ietf-sid-file

Registrant Contact: The IESG.

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

Reference: RFC XXXX

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

7.2. Register ".sid" File Format Module

This document registers one YANG module in the "YANG Module Names" registry [RFC6020]:

* name: ietf-sid-file

* namespace: urn:ietf:params:xml:ns:yang:ietf-sid-file

* prefix: sid

* reference: RFC XXXX

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

7.3. Create new IANA Registry: "YANG SID Mega-Range" registry

The name of this registry is "YANG SID Mega-Range". This registry is used to record the delegation of the management of a block of SIDs to third parties (such as SDOs or registrars).

7.3.1. Structure

Each entry in this registry must include:

- * The entry point (first SID) of the registered SID block.
- * The size of the registered SID block. The size SHOULD be one million (1 000 000) SIDs, it MAY exceptionally be a multiple of 1 000 000.
- * The contact information of the requesting organization including:
 - The policy of SID range allocations: Public, Private or Both.
 - Organization name
 - URL

7.3.2. Allocation policy

The IANA policy for future additions to this registry is "Expert Review" [RFC8126].

An organization requesting to manage a YANG SID Range (and thus have an entry in the YANG SID Mega-Range Registry), must ensure the following capacities:

- * The capacity to manage and operate a YANG SID Range Registry. A YANG SID Range Registry MUST provide the following information for all YANG SID Ranges allocated by the Registry:
 - Entry Point of allocated YANG SID Range
 - Size of allocated YANG SID Range
 - Type: Public or Private
 - o Public Ranges MUST include at least a reference to the YANG module and ".sid" files for that YANG SID Range (e.g., compare Section 7.4.3 for the IETF YANG SID registry).
 - o Private Ranges MUST be marked as "Private"

- * A Policy of allocation, which clearly identifies if the YANG SID Range allocations would be Private, Public or Both.
- * Technical capacity to ensure the sustained operation of the registry for a period of at least 5 years. If Private Registrations are allowed, the period must be of at least 10 years.

If a size of the allocation beyond 1 000 000 is desired, the organization must demonstrate the sustainability of the technical approach for utilizing this size of allocation and how it does not negatively impact the overall usability of the SID allocation mechanisms; such allocations are preferably placed in the space above 4 295 000 000 (64-bit space).

7.3.2.1. First allocation

For a first allocation to be provided, the requesting organization must demonstrate a functional registry infrastructure.

7.3.2.2. Consecutive allocations

On subsequent allocation request(s), the organization must demonstrate the exhaustion of the prior range. These conditions need to be asserted by the assigned expert(s).

If that extra-allocation is done within 3 years from the last allocation, the experts need to discuss this request on the CORE working group mailing list and consensus needs to be obtained before allocating a new Mega-Range.

7.3.3. Initial contents of the Registry

The initial entry in this registry is allocated to IANA:

Entry Point	Size	Allocation	Organization name	URL
0	1000000	Public	IANA	iana.org

Table 1

7.4. Create a new IANA Registry: IETF YANG SID Range Registry (managed by IANA)

7.4.1. Structure

Each entry in this registry must include:

- * The SID range entry point.
- * The SID range size.
- * The YANG module name.
- * Document reference.

7.4.2. Allocation policy

The first million SIDs assigned to IANA is sub-divided as follows:

- * The range of 0 to 999 (size 1000) is subject to "IESG Approval" as defined in [RFC8126]; of these, SID value 0 has been reserved for implementations to internally signify the absence of a SID number and does not occur in interchange.
- * The range of 1000 to 59,999 (size 59,000) is designated for YANG modules defined in RFCs.
 - The IANA policy for additions to this registry is either:
 - o "Expert Review" [RFC8126] in case the ".sid" file comes from a YANG module from an existing RFC, or
 - o "RFC Required" [RFC8126] otherwise.
 - The Expert MUST verify that the YANG module for which this allocation is made has an RFC (existing RFC) OR is on track to become RFC (early allocation with a request from the WG chairs as defined by [BCP100]).
- * The range of 60,000 to 99,999 (size 40,000) is reserved for experimental YANG modules. This range MUST NOT be used in operational deployments since these SIDs are not globally unique which limit their interoperability. The IANA policy for this range is "Experimental use" [RFC8126].
- * The range of 100,000 to 999,999 (size 900,000) is "Reserved" as defined in [RFC8126].

Entry Point	Size	IANA policy
0	1,000	IESG Approval
1,000	59,000	RFC Required
60,000	40,000	Experimental/Private use
100,000	900,000	Reserved

Table 2

The size of the SID range allocated for a YANG module is recommended to be a multiple of 50 and to 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. The SID range size SHOULD NOT exceed 1000; a larger size may be requested by the authors if this recommendation is considered insufficient. It is important to note that an additional SID range can be allocated to an existing YANG module if the initial range is exhausted; this then just leads to slightly less efficient representation.

In case a SID range is allocated for an existing RFC through the "Expert Review" policy, the Document reference field for the given allocation should point to the RFC that the YANG module is defined in.

In case a SID range is required before publishing the RFC due to implementations needing stable SID values, early allocation as defined in [BCP100] can be employed. As specified in Section 4.6 of [RFC8126], RFCs and by extension documents that are expected to become an RFC fulfill the requirement for "Specification Required" stated in Section 2 of [BCP100], which allows for the early allocation process to be employed.

7.4.3. Publication of the ".sid" file

For a YANG module approved for publication as an RFC, a ".sid" file SHOULD be included in the Internet-Draft as a source code block. This ".sid" file is to be extracted by IANA/the expert reviewer and put into the YANG SID Registry (Section 7.5) along with the YANG module. The ".sid" file MUST NOT be published as part of the RFC: the IANA Registry is authoritative and a link is to be inserted in the RFC.

7.4.4. Initial contents of the registry

Initial entries in this registry are as follows:

Entry Point	Size	Module name	Document reference
0	1	(Reserved: not a valid SID)	RFCXXXX
1000	100	ietf-coreconf	[I-D.ietf-core-comi]
1100	50	ietf-yang-types	[RFC6991]
1150	50	ietf-inet-types	[RFC6991]
1200	50	iana-crypt-hash	[RFC7317]
1250	50	ietf-netconf-acm	[RFC8341]
1300	50	ietf-sid-file	RFCXXXX
1500	100	ietf-interfaces	[RFC8343]
1600	100	ietf-ip	[RFC8344]
1700	100	ietf-system	[RFC7317]
1800	400	iana-if-type	[RFC7224]
2400	50	ietf-voucher	[RFC8366]
2450	50	ietf-constrained-voucher	[I-D.ietf-anima-constrained-voucher]
2500	50	ietf-constrained-voucher-request	[I-D.ietf-anima-constrained-voucher]

Table 3

// RFC Ed.: replace XXXX with RFC number assigned to this draft.

For allocation, RFC publication of the YANG module is required as per [RFC8126]. The YANG module must be registered in the "YANG module Name" registry according to the rules specified in Section 14 of [RFC6020].

7.5. Create new IANA Registry: "IETF YANG SID Registry"

The name of this registry is "IETF YANG SID Registry". This registry is used to record the allocation of SIDs for individual YANG module items.

7.5.1. Structure

Each entry in this registry must include:

- * The YANG module name. This module name must be present in the "Name" column of the "YANG Module Names" registry.
- * A link to the associated ".yang" file. This file link must be present in the "File" column of the "YANG Module Names" registry.
- * The link to the ".sid" file which defines the allocation. The ".sid" file is stored by IANA.
- * The number of actually allocated SIDs in the ".sid" file.

7.5.2. Allocation policy

The allocation policy is Expert review. The Expert MUST ensure that the following conditions are met:

- * The ".sid" file has a valid structure:
 - The ".sid" file MUST be a valid JSON file following the structure of the module defined in RFCXXXX (RFC Ed: replace XXX with RFC number assigned to this draft).
- * The ".sid" file allocates individual SIDs ONLY in the YANG SID Ranges for this YANG module (as allocated in the IETF YANG SID Range Registry):
 - All SIDs in this ".sid" file MUST be within the ranges allocated to this YANG module in the "IETF YANG SID Range Registry".
- * If another ".sid" file has already allocated SIDs for this YANG module (e.g. for older or newer versions of the YANG module), the YANG items are assigned the same SIDs as in the other ".sid" file.

- * If there is an older version of the ".sid" file, all allocated SIDs from that version are still present in the current version of the ".sid" file.

7.5.3. Recursive Allocation of YANG SID Range at Document Adoption

Due to the difficulty in changing SID values during IETF document processing, it is expected that most documents will ask for SID allocations using Early Allocations [BCP100]. The details of the Early Allocation should be included in any Working Group Adoption call. Prior to Working Group Adoption, an internet draft author can use the experimental SID range (as per Section 7.4.2) for their SIDs allocations or other values that do not create ambiguity with other SID uses (for example they can use a range that comes from a non-IANA managed "YANG SID Mega-Range" registry).

After Working Group Adoption, any modification of a ".sid" file is expected to be discussed on the mailing list of the appropriate Working Groups. Specific attention should be paid to implementers' opinion after Working Group Last Call if a SID value is to change its meaning. In all cases, a ".sid" file and the SIDs associated with it are subject to change before the publication of an internet draft as an RFC.

During the early use of SIDs, many existing, previously published YANG modules will not have SID allocations. For an allocation to be useful the included YANG modules may also need to have SID allocations made.

The Expert Reviewer who performs the (Early) Allocation analysis will need to go through the list of included YANG modules and perform SID allocations for those modules as well.

- * If the document is a published RFC, then the allocation of SIDs for its referenced YANG modules is permanent. The Expert Reviewer provides the generated ".sid" file to IANA for registration. This process may be time-consuming during a bootstrap period (there are over 100 YANG modules to date, none of which have SID allocations), but should quiet down once needed entries are allocated.
- * If the document is an unprocessed Internet-Draft adopted in a WG, then an Early Allocation is performed for this document as well. Early Allocations require approval by an IESG Area Director. An early allocation which requires additional allocations will list the other allocations in its description, and will be cross-posted to the any other working group mailing lists.

- * A YANG module which references a module in a document which has not yet been adopted by any working group will be unable to perform an Early Allocation for that other document until it is adopted by a working group. As described in [BCP100], an AD Sponsored document acts as if it had a working group. The approving AD may also exempt a document from this policy by agreeing to AD Sponsor the document.

At the end of the IETF process all the dependencies of a given module for which SIDs are assigned, should also have SIDs assigned. Those dependencies' assignments should be permanent (not Early Allocation).

A previously SID-allocated YANG module which changes its references to include a YANG module for which there is no SID allocation needs to repeat the Early Allocation process.

Early Allocations are made with a one-year period, after which they are expired. [BCP100] indicates that at most one renewal may be made. For the SID allocation a far more lenient stance is desired.

1. An extension of a referencing documents Early Allocation should update any referenced Early Allocations to expire no sooner than the referencing document.
2. The [BCP100] mechanism allows the IESG to provide a second renewal, and such an event may prompt some thought about how the collection of documents are being processed.

This is driven by the very generous size of the SID space and the often complex and deep dependencies of YANG modules. Often a core module with many dependencies will undergo extensive review, delaying the publication of other documents.

[BCP100] also says:

Note that if a document is submitted for review to the IESG and at the time of submission some early allocations are valid (not expired), these allocations should not be expired while the document is under IESG consideration or waiting in the RFC Editor's queue after approval by the IESG.

7.5.4. Initial contents of the registry

None.

8. References

8.1. Normative References

- [BCP100] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, January 2014.

<<https://www.rfc-editor.org/info/bcp100>>

- [I-D.ietf-core-yang-cbor]

Veillette, M., Petrov, I., Pelov, A., Bormann, C., and M. Richardson, "CBOR Encoding of Data Modeled with YANG", Work in Progress, Internet-Draft, draft-ietf-core-yang-cbor-17, 25 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-yang-cbor-17.txt>>.

- [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>>.

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

- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.

- [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>>.

- [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>>.

- [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>>.

8.2. Informative References

- [I-D.ietf-anima-constrained-voucher]
Richardson, M., Stok, P. V. D., Kampanakis, P., and E. Dijk, "Constrained Bootstrapping Remote Secure Key Infrastructure (BRSKI)", Work in Progress, Internet-Draft, draft-ietf-anima-constrained-voucher-14, 25 October 2021, <<https://www.ietf.org/archive/id/draft-ietf-anima-constrained-voucher-14.txt>>.
- [I-D.ietf-core-comi]
Veillette, M., Stok, P. V. D., Pelov, A., Bierman, A., and I. Petrov, "CoAP Management Interface (CORECONF)", Work in Progress, Internet-Draft, draft-ietf-core-comi-11, 17 January 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-comi-11.txt>>.
- [I-D.ietf-core-yang-library]
Veillette, M. and I. Petrov, "Constrained YANG Module Library", Work in Progress, Internet-Draft, draft-ietf-core-yang-library-03, 11 January 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-yang-library-03.txt>>.
- [PYANG] Bjorklund, M., "An extensible YANG validator and converter in python", <<https://github.com/mbj4668/pyang>>.
- [RFC6020] Bjorklund, M., Ed., "YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF)", RFC 6020, DOI 10.17487/RFC6020, October 2010, <<https://www.rfc-editor.org/info/rfc6020>>.
- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC7224] Bjorklund, M., "IANA Interface Type YANG Module", RFC 7224, DOI 10.17487/RFC7224, May 2014, <<https://www.rfc-editor.org/info/rfc7224>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <<https://www.rfc-editor.org/info/rfc7317>>.

- [RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <<https://www.rfc-editor.org/info/rfc8126>>.
- [RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <<https://www.rfc-editor.org/info/rfc8341>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.
- [RFC8344] Bjorklund, M., "A YANG Data Model for IP Management", RFC 8344, DOI 10.17487/RFC8344, March 2018, <<https://www.rfc-editor.org/info/rfc8344>>.
- [RFC8366] Watsen, K., Richardson, M., Pritikin, M., and T. Eckert, "A Voucher Artifact for Bootstrapping Protocols", RFC 8366, DOI 10.17487/RFC8366, May 2018, <<https://www.rfc-editor.org/info/rfc8366>>.

Appendix A. ".sid" file example

The following ".sid" file (ietf-system@2014-08-06.sid) has been generated using the following yang modules:

- * ietf-system@2014-08-06.yang (defined in [RFC7317])
- * ietf-yang-types@2013-07-15.yang (defined in [RFC6991])
- * ietf-inet-types@2013-07-15.yang (defined in [RFC6991])
- * ietf-netconf-acm@2018-02-14.yang (defined in [RFC8341])
- * iana-crypt-hash@2014-08-06.yang (defined in [RFC7317])

For purposes of exposition, line breaks have been introduced below in some JSON strings that represent overly long identifiers.

```
{
  "ietf-sid-file:sid-file" : {
    "module-name": "ietf-system",
    "module-revision": "2014-08-06",
    "dependency-revision": [
      {
        "module-name": "ietf-yang-types",
```

```
    "module-revision": "2013-07-15"
  },
  {
    "module-name": "ietf-inet-types",
    "module-revision": "2013-07-15"
  },
  {
    "module-name": "ietf-netconf-acm",
    "module-revision": "2018-02-14"
  },
  {
    "module-name": "iana-crypt-hash",
    "module-revision": "2014-08-06"
  }
],
"description": "Example sid file",
"assignment-range": [
  {
    "entry-point": 1700,
    "size": 100
  }
],
"item": [
  {
    "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
  }
]
}
}

```

Figure 3: Example .sid file (ietf-system, with extra line-breaks)

For reconstructing the actual JSON file from this figure, all line breaks that occur in what would be JSON strings need to be removed, including any following blank space (indentation) on the line after the line break; in each such case, a single identifier without any embedded blank space results. This removal can be accomplished with this simple Ruby script:

```

@u = %{[^\n]*}; @q = @u + "'"
puts ARGF.read.gsub(/^(#@q#@q#@q)*#@u) *\n + (#@q)/, "\\1\\3")

```

Appendix B. SID auto generation

Assignment of SIDs to YANG items SHOULD be automated. The recommended process to assign SIDs is as follows:

1. A tool extracts the different items defined for a specific YANG module.
2. The list of items is sorted in alphabetical order, 'namespace' in descending order, 'identifier' in ascending order. The 'namespace' and 'identifier' formats are described in the YANG module 'ietf-sid-file' defined in Section 4.
3. SIDs are assigned sequentially from the entry point up to the size of the registered SID range. This approach is recommended to minimize the serialization overhead, especially when delta between a reference SID and the current SID is used by protocols aiming to reduce message size.
4. If the number of items exceeds the SID range(s) allocated to a YANG module, an extra range is added for subsequent assignments.
5. The "dependency-revision" should reflect the revision numbers of each YANG module that the YANG module imports at the moment of the generation.

When updating a YANG module that is in active use, the existing SID assignments are maintained. (In contrast, when evolving an early draft that has not yet been adopted by a community of developers, SID assignments are often better done from scratch after a revision.) If the name of a schema node changes, but the data remain structurally and semantically similar to what was previously available under an old name, the SID that was used for the old name MAY continue to be used for the new name. If the meaning of an item changes, a new SID MAY be assigned to it; this is particularly useful to allow the new SID to identify the new structure or semantics of the item. If the YANG data type changes in a new revision of a published module, such that the resulting CBOR encoding is changed, then implementations will be aided significantly if a new SID is assigned. Note that these decisions are generally at the discretion of the YANG module author, who should decide if the benefits of a manual intervention are worth the deviation from automatic assignment.

In case of an update to an existing ".sid" file, an additional step is needed that increments the ".sid" file version number. If there was no version number in the previous version of the ".sid" file, 0 is assumed as the version number of the old version of the ".sid" file and the version number is 1 for the new ".sid" file. Apart from

that, changes of ".sid" files can also be automated using the same method described above, only unassigned YANG items are processed at step #3. Already existing items in the ".sid" file should not be given new SIDs.

Note that ".sid" file versions are specific to a YANG module revision. For each new YANG module or each new revision of an existing YANG module, the version number of the initial ".sid" file should either be 0 or should not be present.

Note also that RPC or action "input" and "output" data nodes MUST always be assigned SID even if they don't contain data nodes. The reason for this requirement is that other modules can augment the given module and those SIDs might be necessary.

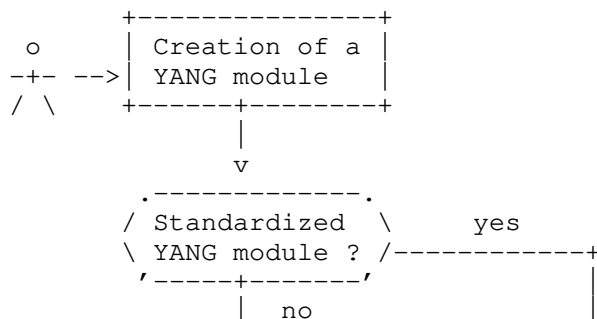
Appendix C. ".sid" file lifecycle

Before assigning SIDs to their YANG modules, YANG module authors must acquire a SID range from a "YANG SID Range Registry". If the YANG module is part of an IETF draft or RFC, the SID range need to be acquired from the "IETF YANG SID Range Registry" as defined in Section 7.4. For the other YANG modules, the authors can acquire a SID range from any "YANG SID Range Registry" of their choice.

Once the SID range is acquired, owners can use it to generate ".sid" file/s for their YANG module/s. It is recommended to leave some unallocated SIDs following the allocated range in each ".sid" file in order to allow better evolution of the YANG module in the future. Generation of ".sid" files should be performed using an automated tool. Note that ".sid" files can only be generated for YANG modules and not for submodules.

C.1. ".sid" File Creation

The following activity diagram summarizes the creation of a YANG module and its associated ".sid" file.



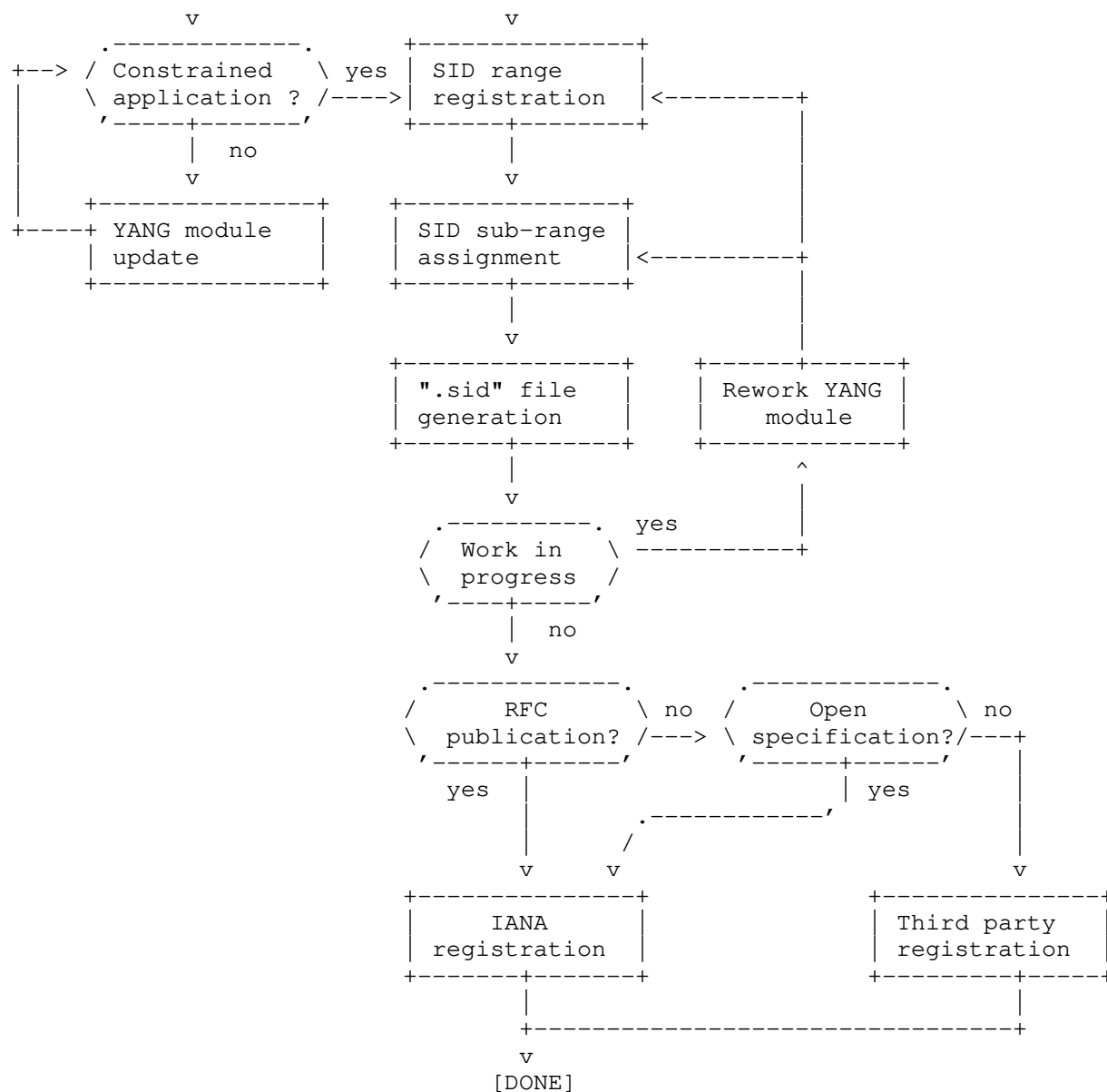


Figure 4: SID Lifecycle

C.2. ".sid" File Update

The following Activity diagram summarizes the update of a YANG module and its associated ".sid" file.

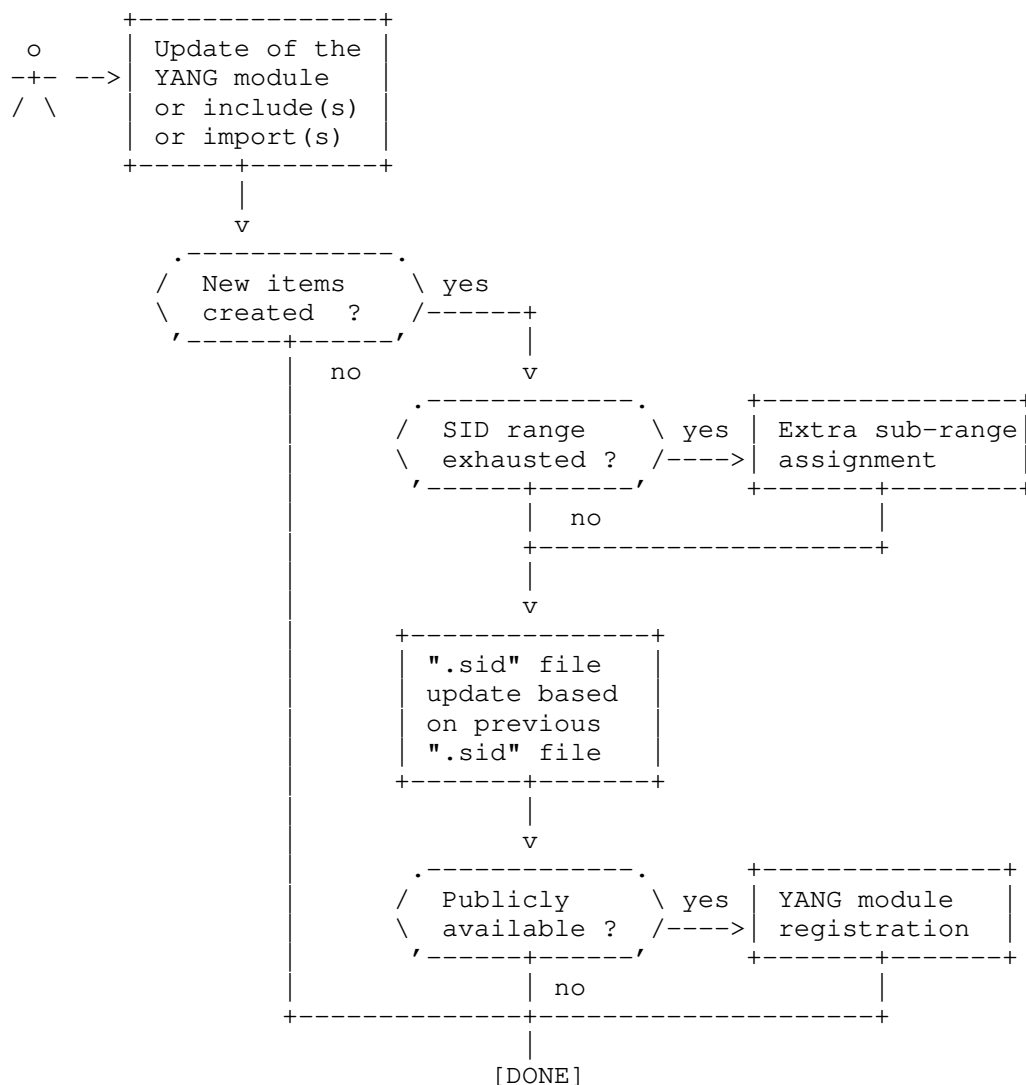


Figure 5: YANG and ".sid" file update

Acknowledgments

The authors would like to thank Andy Bierman, Michael Richardson, Abhinav Somaraju, Peter van der Stok, Laurent Toutain and Randy Turner for their help during the development of this document and their useful comments during the review process.

Contributors

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

Email: andy@yumaworks.com

Authors' Addresses

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

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

Alexander Pelov (editor)
Acklio
1137A avenue des Champs Blancs
35510 Cesson-Sevigne
France

Email: a@ackl.io

Ivaylo Petrov (editor)
Google Switzerland GmbH
Brandschenkestrasse 110
CH-8002 Zurich
Switzerland

Email: ivaylopetrov@google.com

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

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

Michael Richardson
Sandelman Software Works

Email: mcr+ietf@sandelman.ca

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: 13 October 2022

M. Veillette, Ed.
Trilliant Networks Inc.
I. Petrov, Ed.
Google Switzerland GmbH
A. Pelov
Acklio
C. Bormann
Universität Bremen TZI
M. Richardson
Sandelman Software Works
11 April 2022

CBOR Encoding of Data Modeled with YANG
draft-ietf-core-yang-cbor-20

Abstract

Based on the Concise Binary Object Representation (CBOR, RFC 8949), this document defines encoding rules for representing configuration data, state data, parameters and results of Remote Procedure Call (RPC) operations or actions, and notifications, defined using YANG (RFC 7950).

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 13 October 2022.

Copyright Notice

Copyright (c) 2022 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 Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. Terminology and Notation	3
3. Properties of the CBOR Encoding	5
3.1. CBOR diagnostic notation	6
3.2. YANG Schema Item iDentifier	8
3.3. Name	9
4. Encoding of Representation Nodes	11
4.1. The 'leaf'	11
4.1.1. Using SIDs in keys	11
4.1.2. Using names in keys	12
4.2. The 'container' and other nodes from the data tree	12
4.2.1. Using SIDs in keys	13
4.2.2. Using names in keys	14
4.3. The 'leaf-list'	15
4.3.1. Using SIDs in keys	15
4.3.2. Using names in keys	16
4.4. The 'list' and 'list' entries	16
4.4.1. Using SIDs in keys	17
4.4.2. Using names in keys	19
4.5. The 'anydata'	21
4.5.1. Using SIDs in keys	22
4.5.2. Using names in keys	23
4.6. The 'anyxml'	24
4.6.1. Using SIDs in keys	24
4.6.2. Using names in keys	25
5. Encoding of 'yang-data' extension	25
5.1. Using SIDs in keys	26
5.2. Using names in keys	27
6. Representing YANG Data Types in CBOR	28
6.1. The unsigned integer Types	28
6.2. The integer Types	29
6.3. The 'decimal64' Type	29
6.4. The 'string' Type	30
6.5. The 'boolean' Type	30
6.6. The 'enumeration' Type	31
6.7. The 'bits' Type	32
6.8. The 'binary' Type	34

6.9. The 'leafref' Type	35
6.10. The 'identityref' Type	35
6.10.1. SIDs as identityref	35
6.10.2. Name as identityref	36
6.11. The 'empty' Type	37
6.12. The 'union' Type	37
6.13. The 'instance-identifier' Type	38
6.13.1. SIDs as instance-identifier	39
6.13.2. Names as instance-identifier	42
7. Content-Types	43
8. Security Considerations	44
9. IANA Considerations	45
9.1. Media-Types Registry	45
9.2. CoAP Content-Formats Registry	46
9.3. CBOR Tags Registry	46
10. References	47
10.1. Normative References	47
10.2. Informative References	48
Acknowledgments	49
Authors' Addresses	49

1. Introduction

The specification of the YANG 1.1 data modeling language [RFC7950] defines an XML encoding for data instances, i.e., contents of configuration datastores, state data, RPC inputs and outputs, action inputs and outputs, and event notifications.

An additional set of encoding rules has been defined in [RFC7951] based on the JavaScript Object Notation (JSON) Data Interchange Format [RFC8259].

The aim of this document is to define a set of encoding rules for the Concise Binary Object Representation (CBOR) [RFC8949], collectively called `_YANG-CBOR_`. The resulting encoding is more compact compared to XML and JSON and more suitable for Constrained Nodes and/or Constrained Networks as defined by [RFC7228].

2. Terminology and Notation

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

The following terms are defined in [RFC7950]:

- * action
- * anydata
- * anyxml
- * data node
- * data tree
- * datastore
- * feature
- * identity
- * module
- * notification
- * RPC
- * schema node
- * submodule

The following term is defined in [RFC8040]:

- * yang-data extension

The following term is defined in [RFC8791]:

- * YANG data structure

This specification also makes use of the following terminology:

- * YANG Schema Item iDentifier (YANG SID or simply SID): 63-bit unsigned integer used to identify different YANG items.
- * delta: Difference between the current YANG SID and a reference YANG SID. A reference YANG SID is defined for each context for which deltas are used.
- * absolute SID: YANG SID not encoded as a delta. This is usually called out explicitly only in positions where normally a delta would be found.

- * **representation tree**: a YANG data tree, possibly enclosed by a representation of a schema node such as a YANG data structure, a notification, an RPC, or an action.
- * **representation node**: a node in a representation tree, i.e., a data tree node, or a representation of a schema node such as a YANG data structure, a notification, an RPC, or an action.
- * **item**: A schema node, an identity, a module, or a feature defined using the YANG modeling language.
- * **list entry**: the data associated with a single entry of a list (see Section 7.8 of [RFC7950]).
- * **parent (of a representation node)**: the schema node of the closest enclosing representation node in which a given representation node is defined.

3. Properties of the CBOR Encoding

This document defines CBOR encoding rules for YANG data trees and their subtrees.

A YANG data tree can be enclosed by a representation of a schema node such as a YANG data structure, a notification, an RPC, or an action; this is called a representation tree. The data tree nodes and the enclosing schema node representation, if any, are collectively called the representation nodes.

A representation node such as container, list entry, YANG data structure, notification, RPC input, RPC output, action input, or action output is serialized using a CBOR map in which each schema node defined within is encoded using a key and a value. This specification supports two types of CBOR keys; YANG Schema Item identifier (YANG SID) as defined in Section 3.2 and names as defined in Section 3.3. Each of these key types is encoded using a specific CBOR type which allows their interpretation during the deserialization process. Protocols or mechanisms implementing this specification can mandate the use of a specific key type or allow the generator to choose freely per key.

In order to minimize the size of the encoded data, the mapping avoids any unnecessary meta-information beyond that directly provided by the CBOR basic generic data model (Section 2 of [RFC8949]). For instance, CBOR tags are used solely in the case of an absolute SID, anyxml data nodes, or the union datatype, to distinguish explicitly the use of different YANG datatypes encoded using the same CBOR major type.

Unless specified otherwise by the protocol or mechanism implementing this specification, the indefinite length encoding as defined in Section 3.2 of [RFC8949] SHALL be supported by the CBOR decoders employed with YANG-CBOR. (This enables an implementation to begin emitting an array or map before the number of entries in that structure is known, possibly also avoiding excessive locking or race conditions. On the other hand, it deprives the receiver of the encoded data from advance announcement about some size information, so a generator should choose indefinite length encoding only when these benefits do accrue.)

Data nodes implemented using a CBOR array, map, byte string, or text string can be instantiated but empty. In this case, they are encoded with a length of zero.

When representation nodes are serialized using the rules defined by this specification as part of an application payload, the payload SHOULD include information that would allow a stateless way to identify each node, such as the SID number associated with the node, SID delta from another SID in the application payload, the namespace qualified name, or the instance-identifier.

Examples in Section 4 include a root CBOR map with a single entry having a key set to either a namespace qualified name or a SID. This root CBOR map is provided only as a typical usage example and is not part of the present encoding rules. Only the value within this CBOR map is compulsory.

3.1. CBOR diagnostic notation

Within this document, CBOR binary contents are represented using an equivalent textual form called CBOR diagnostic notation as defined in Section 8 of [RFC8949]. This notation is used strictly for documentation purposes and is never used in the data serialization. Table 1 below provides a summary of this notation.

CBOR content	CBOR type	Diagnostic notation	Example	CBOR encoding
Unsigned integer	0	Decimal digits	123	18 7B
Negative integer	1	Decimal digits prefixed by a minus sign	-123	38 7A
Byte string	2	Hexadecimal value enclosed between single quotes and prefixed by an 'h'	h'F15C'	42 F15C
Text string	3	String of Unicode characters enclosed between double quotes	"txt"	63 747874
Array	4	Comma-separated list of values within square brackets	[1, 2]	82 01 02
Map	5	Comma-separated list of key : value pairs within curly braces	{ 1: 123, 2: 456 }	A2 01187B 021901C8
Boolean	7/20	false	false	F4
	7/21	true	true	F5
Null	7/22	null	null	F6
Not assigned	7/23	undefined	undefined	F7

Table 1: CBOR diagnostic notation summary

Note: CBOR binary contents shown in this specification are annotated with comments. These comments are delimited by slashes ("/") as defined in [RFC8610] Appendix G.6.

3.2. YANG Schema Item iDentifier

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both Network Configuration Protocol (NETCONF) [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using text strings. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG items is required. This compact identifier, called YANG Schema Item iDentifier, is an unsigned integer limited to 63 bits of range (i.e., 0..9223372036854775807 or 0..0xffffffffffffffff). The following items are identified using YANG SIDs (often shortened to SIDs):

- * identities
- * data nodes
- * RPCs and associated input(s) and output(s)
- * actions and associated input(s) and output(s)
- * YANG data structures
- * notifications and associated information
- * YANG modules and features

Note that any structuring of modules into submodules is transparent to YANG-CBOR: SIDs are not allocated for the names of submodules, and any items within a submodule are effectively allocated SIDs as part of processing the module that includes them.

To minimize their size, SIDs used as keys in CBOR maps are encoded using deltas, i.e., signed (negative or unsigned) integers that are added to the reference SID applying to the map. The reference SID of an outermost map is zero, unless a different reference SID is unambiguously conferred from the environment in which the outermost map is used. The reference SID of a map that is most directly embedded in a map entry with a name-based key is zero. For all other maps, the reference SID is the SID computed for the map entry it is most directly embedded in. (The embedding may be indirect if an array intervenes, e.g., in a YANG list.) Where absolute SIDs are desired in map key positions (where a bare integer implies a delta), they need to be identified as absolute SID values by using CBOR tag number 47 (as defined in Section 4.2.1).

Thus, conversion from SIDs to deltas and back to SIDs is a stateless process solely based on the data serialized or deserialized combined with, potentially, an outermost reference SID unambiguously conferred by the environment.

Mechanisms and processes used to assign SIDs to YANG items and to guarantee their uniqueness are outside the scope of the present specification. If SIDs are to be used, the present specification is used in conjunction with a specification defining this management. A related document, [I-D.ietf-core-sid], is intended to serve as the definitive way to assign SID values for YANG modules managed by the IETF, and recommends itself for YANG modules managed by non-IETF entities, as well. The present specification has been designed to allow different methods of assignment to be used within separate domains.

To provide implementations with a way to internally indicate the absence of a SID, the SID value 0 is reserved and will not be allocated; it is not used in interchange.

3.3. Name

This specification also supports the encoding of YANG item identifiers as text strings, similar to those used by the JSON Encoding of Data Modeled with YANG [RFC7951]. This approach can be used to avoid the management overhead associated with SID allocation. The main drawback is the significant increase in size of the encoded data.

YANG item identifiers implemented using names MUST be in one of the following forms:

- * simple -- the identifier of the YANG item (i.e., schema node or identity).
- * namespace qualified -- the identifier of the YANG item is prefixed with the name of the module in which this item is defined, separated by the colon character (":").

The name of a module determines the namespace of all YANG items defined in that module. If an item is defined in a submodule, then the namespace qualified name uses the name of the main module to which the submodule belongs.

ABNF syntax [RFC5234] of a name is shown in Figure 1, where the production for "identifier" is defined in Section 14 of [RFC7950].

```
name = [identifier ":" ] identifier
```

Figure 1: ABNF Production for a simple or namespace qualified name

A namespace qualified name MUST be used for all members of a top-level CBOR map and then also whenever the namespaces of the representation node and its parent node are different. In all other cases, the simple form of the name MUST be used.

Definition example:

```
module example-foomod {
  container top {
    leaf foo {
      type uint8;
    }
  }
}

module example-barmod {
  import example-foomod {
    prefix "foomod";
  }
  augment "/foomod:top" {
    leaf bar {
      type boolean;
    }
  }
}
```

A valid CBOR encoding of the 'top' container is as follows.

CBOR diagnostic notation:

```
{
  "example-foomod:top": {
    "foo": 54,
    "example-barmod:bar": true
  }
}
```

Both the 'top' container and the 'bar' leaf defined in a different YANG module as its parent container are encoded as namespace qualified names. The 'foo' leaf defined in the same YANG module as its parent container is encoded as simple name.

4. Encoding of Representation Nodes

Representation nodes defined using the YANG modeling language are encoded using CBOR [RFC8949] based on the rules defined in this section. We assume that the reader is already familiar with both YANG [RFC7950] and CBOR [RFC8949].

4.1. The 'leaf'

A 'leaf' MUST be encoded accordingly to its datatype using one of the encoding rules specified in Section 6.

The following examples show the encoding of a 'hostname' leaf using a SID or a name.

Definition example adapted from [RFC6991] and [RFC7317]:

```
typedef domain-name {
  type string {
    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]\.?)'
      + '\.?' ;
    length "1..253";
  }
}

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

4.1.1. Using SIDs in keys

As with all examples below, the delta in the outermost map assumes a reference YANG SID (current schema node) of 0.

CBOR diagnostic notation:

```
{
  1752 : "myhost.example.com"      / hostname (SID 1752) /
}
```

CBOR encoding:

```
A1                                # map(1)
  19 06D8                          # unsigned(1752)
  72                               # text(18)
    6D79686F737442E6578616D706C652E636F6D # "myhost.example.com"
```

4.1.2. Using names in keys

CBOR diagnostic notation:

```
{
  "ietf-system:hostname" : "myhost.example.com"
}
```

CBOR encoding:

```
A1                                # map(1)
  74                              # text(20)
    6965746662D73797374656D3A686F73746E616D65
  72                              # text(18)
    6D79686F737442E6578616D706C652E636F6D
```

4.2. The 'container' and other nodes from the data tree

Instances of containers, YANG data structures, notification contents, RPC inputs, RPC outputs, action inputs, and action outputs MUST be encoded using a CBOR map data item (major type 5). The same encoding is also used for the list entries in a list (Section 4.4). A map consists of pairs of data items, with each pair consisting of a key and a value. Each key within the CBOR map is set to a schema node identifier, each value is set to the value of this representation node according to the instance datatype.

This specification supports two types of CBOR map keys; SID as defined in Section 3.2 and names as defined in Section 3.3.

The following examples show the encoding of a 'system-state' container representation instance using SIDs or names.

Definition example adapted from [RFC6991] and [RFC7317]:

```
typedef date-and-time {  
  type string {  
    pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?'  
      + '(Z|[\+|-]\d{2}:\d{2})';  
  }  
}  
  
container system-state {  
  
  container clock {  
    leaf current-datetime {  
      type date-and-time;  
    }  
  
    leaf boot-datetime {  
      type date-and-time;  
    }  
  }  
}
```

4.2.1. Using SIDs in keys

In the context of containers and other nodes from the data tree, CBOR map keys within inner CBOR maps can be encoded using deltas (bare integers) or absolute SIDs (tagged with tag number 47).

Delta values are computed as follows:

- * In the case of a 'container', deltas are equal to the SID of the current representation node minus the SID of the parent 'container'.
- * In the case of a 'list', deltas are equal to the SID of the current representation node minus the SID of the parent 'list'.
- * In the case of an 'RPC input' or 'RPC output', deltas are equal to the SID of the current representation node minus the SID of the 'RPC'.
- * In the case of an 'action input' or 'action output', deltas are equal to the SID of the current representation node minus the SID of the 'action'.
- * In the case of a 'notification content', deltas are equal to the SID of the current representation node minus the SID of the 'notification'.

CBOR diagnostic notation:

```

{
  1720 : {
    1 : {
      2 : "2015-10-02T14:47:24Z-05:00", / current-datetime(SID 1723)/
      1 : "2015-09-15T09:12:58Z-05:00" / boot-datetime (SID 1722) /
    }
  }
}

```

CBOR encoding:

```

A1                                # map(1)
19 06B8                          # unsigned(1720)
A1                                # map(1)
  01                             # unsigned(1)
  A2                             # map(2)
    02                          # unsigned(2)
    78 1A                       # text(26)
      323031352D31302D30325431343A34373A32345A2D30353A3030
    01                          # unsigned(1)
    78 1A                       # text(26)
      323031352D30392D31355430393A31323A35385A2D30353A3030

```

Figure 2: System state clock encoding

4.2.2. Using names in keys

CBOR map keys implemented using names MUST be encoded using a CBOR text string data item (major type 3). A namespace-qualified name MUST be used each time the namespace of a representation node and its parent differ. In all other cases, the simple form of the name MUST be used. Names and namespaces are defined in Section 4 of [RFC7951].

The following example shows the encoding of a 'system' container representation node instance using names.

CBOR diagnostic notation:

```

{
  "ietf-system:system-state" : {
    "clock" : {
      "current-datetime" : "2015-10-02T14:47:24Z-05:00",
      "boot-datetime" : "2015-09-15T09:12:58Z-05:00"
    }
  }
}

```

CBOR encoding:

```

A1                                     # map(1)
  78 18                               # text(24)
    696574662D73797374656D3A73797374656D2D7374617465
A1                                     # map(1)
  65                                  # text(5)
    636C6F636B                       # "clock"
A2                                     # map(2)
  70                                  # text(16)
    63757272656E742D6461746574696D65
  78 1A                               # text(26)
    323031352D31302D30325431343A34373A32345A2D30353A3030
  6D                                  # text(13)
    626F6F742D6461746574696D65
  78 1A                               # text(26)
    323031352D30392D31355430393A31323A35385A2D30353A3030

```

4.3. The 'leaf-list'

A leaf-list MUST be encoded using a CBOR array data item (major type 4). Each entry of this array MUST be encoded accordingly to its datatype using one of the encoding rules specified in Section 6.

The following example shows the encoding of the 'search' leaf-list representation node instance containing two entries, "ietf.org" and "ieee.org".

Definition example adapted from [RFC6991] and [RFC7317]:

```

typedef domain-name {
  type string {
    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]\.?)'
      + '|\.';
    length "1..253";
  }
}

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

```

4.3.1. Using SIDs in keys

CBOR diagnostic notation:

```
{
  1746 : [ "ietf.org", "ieee.org" ]      / search (SID 1746) /
}
```

CBOR encoding:

```
A1                                # map(1)
  19 06D2                        # unsigned(1746)
  82                              # array(2)
    68                          # text(8)
    696574662E6F7267            # "ietf.org"
    68                          # text(8)
    696565652E6F7267            # "ieee.org"
```

4.3.2. Using names in keys

CBOR diagnostic notation:

```
{
  "ietf-system:search" : [ "ietf.org", "ieee.org" ]
}
```

CBOR encoding:

```
A1                                # map(1)
  72                              # text(18)
  696574662D73797374656D3A736561726368 # "ietf-system:search"
  82                              # array(2)
    68                          # text(8)
    696574662E6F7267            # "ietf.org"
    68                          # text(8)
    696565652E6F7267            # "ieee.org"
```

4.4. The 'list' and 'list' entries

A list or a subset of a list MUST be encoded using a CBOR array data item (major type 4). Each list entry within this CBOR array is encoded using a CBOR map data item (major type 5) based on the encoding rules of a collection as defined in Section 4.2.

It is important to note that this encoding rule also applies to a 'list' representation node instance that has a single entry.

The following examples show the encoding of a 'server' list using SIDs or names.

Definition example simplified from [RFC7317]:


```
list server {
  key name;

  leaf name {
    type string;
  }
  choice transport {
    case udp {
      container udp {
        leaf address {
          type host;
          mandatory true;
        }
        leaf port {
          type port-number;
        }
      }
    }
  }
  leaf association-type {
    type enumeration {
      enum server;
      enum peer;
      enum pool;
    }
    default server;
  }
  leaf iburst {
    type boolean;
    default false;
  }
  leaf prefer {
    type boolean;
    default false;
  }
}
```

4.4.1. Using SIDs in keys

The encoding rules of each 'list' entry are defined in Section 4.2.1.

CBOR diagnostic notation:

```
{
  1756 : [
    {
      3 : "NRC TIC server",      / name (SID 1759) /
      5 : {
        1 : "tic.nrc.ca",      / address (SID 1762) /
        2 : 123                / port (SID 1763) /
      },
      1 : 0,                    / association-type (SID 1757) /
      2 : false,                / iburst (SID 1758) /
      4 : true                   / prefer (SID 1760) /
    },
    {
      3 : "NRC TAC server",      / name (SID 1759) /
      5 : {
        1 : "tac.nrc.ca"        / address (SID 1762) /
      }
    }
  ]
}
```

CBOR encoding:

```

A1                                     # map(1)
  19 06DC                             # unsigned(1756)
  82                                   # array(2)
    A5                                # map(5)
      03                              # unsigned(3)
      6E                              # text(14)
        4E52432054494320736572766572 # "NRC TIC server"
      05                              # unsigned(5)
      A2                              # map(2)
        01                            # unsigned(1)
        6A                            # text(10)
          74696332E6E72632E6361      # "tic.nrc.ca"
        02                            # unsigned(2)
        18 7B                        # unsigned(123)
      01                              # unsigned(1)
      00                              # unsigned(0)
      02                              # unsigned(2)
      F4                              # primitive(20)
      04                              # unsigned(4)
      F5                              # primitive(21)
    A2                                # map(2)
      03                              # unsigned(3)
      6E                              # text(14)
        4E52432054414320736572766572 # "NRC TAC server"
      05                              # unsigned(5)
      A1                              # map(1)
        01                            # unsigned(1)
        6A                            # text(10)
          74616332E6E72632E6361      # "tac.nrc.ca"

```

4.4.2. Using names in keys

The encoding rules of each 'list' entry are defined in Section 4.2.2.

CBOR diagnostic notation:

```
{
  "ietf-system:server" : [
    {
      "name" : "NRC TIC server",
      "udp" : {
        "address" : "tic.nrc.ca",
        "port" : 123
      },
      "association-type" : 0,
      "iburst" : false,
      "prefer" : true
    },
    {
      "name" : "NRC TAC server",
      "udp" : {
        "address" : "tac.nrc.ca"
      }
    }
  ]
}
```

CBOR encoding:

```

A1                                     # map(1)
  72                                 # text(18)
    6965746662D73797374656D3A736572766572
82                                     # array(2)
  A5                                 # map(5)
    64                               # text(4)
      6E616D65                       # "name"
    6E                               # text(14)
      4E52432054494320736572766572
    63                               # text(3)
      756470                         # "udp"
  A2                                 # map(2)
    67                               # text(7)
      61646472657373               # "address"
    6A                               # text(10)
      74696332E6E72632E6361       # "tic.nrc.ca"
    64                               # text(4)
      706F7274                     # "port"
    18 7B                           # unsigned(123)
    70                               # text(16)
      6173736F63696174696F6E2D74797065
    00                               # unsigned(0)
    66                               # text(6)
      696275727374                 # "iburst"
  F4                                 # primitive(20)
    66                               # text(6)
      707265666572                 # "prefer"
  F5                                 # primitive(21)
  A2                                 # map(2)
    64                               # text(4)
      6E616D65                       # "name"
    6E                               # text(14)
      4E52432054414320736572766572
    63                               # text(3)
      756470                         # "udp"
  A1                                 # map(1)
    67                               # text(7)
      61646472657373               # "address"
    6A                               # text(10)
      74616332E6E72632E6361       # "tac.nrc.ca"

```

4.5. The 'anydata'

An anydata serves as a container for an arbitrary set of representation nodes that otherwise appear as normal YANG-modeled data. An anydata representation node 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:

- * CBOR map keys of any inner representation nodes MUST be set to valid deltas or names.
- * CBOR arrays MUST contain either unique scalar values (as a leaf-list, see Section 4.3), or maps (as a list, see Section 4.4).
- * CBOR map values MUST follow the encoding rules of one of the datatypes listed in Section 4.

The following example shows a possible use of an anydata. In this example, an anydata is used to define a representation node containing a notification event; this representation node can be part of a YANG list to create an event logger.

Definition example:

```
module event-log {  
  ...  
  anydata last-event;          # SID 60123  
}
```

This example also assumes the assistance of the following notification.

```
module example-port {  
  ...  
  
  notification example-port-fault { # SID 60200  
    leaf port-name {                # SID 60201  
      type string;  
    }  
    leaf port-fault {                # SID 60202  
      type string;  
    }  
  }  
}
```

4.5.1. Using SIDs in keys

CBOR diagnostic notation:

```

{
  60123 : {
    77 : {
      1 : "0/4/21",
      2 : "Open pin 2"
    }
  }
}

```

CBOR encoding:

```

A1                                # map(1)
  19 EADB                        # unsigned(60123)
    A1                            # map(1)
      18 4D                      # unsigned(77)
        A2                      # map(2)
          01                    # unsigned(1)
          66                    # text(6)
            302F342F3231        # "0/4/21"
          02                    # unsigned(2)
          6A                    # text(10)
            4F70656E2070696E2032 # "Open pin 2"

```

In some implementations, it might be simpler to use the absolute SID encoding (tag number 47) for the anydata root element. CBOR diagnostic notation:

```

{
  60123 : {
    47(60200) : {
      1 : "0/4/21",
      2 : "Open pin 2"
    }
  }
}

```

4.5.2. Using names in keys

CBOR diagnostic notation:

```

{
  "event-log:last-event" : {
    "example-port:example-port-fault" : {
      "port-name" : "0/4/21",
      "port-fault" : "Open pin 2"
    }
  }
}

```

CBOR encoding:

```

A1                                     # map(1)
  74                                 # text(20)
    6576656E742D6C6F673A6C6173742D6576656E74
A1                                     # map(1)
  78 1F                             # text(31)
    6578616D706C652D706F72743A
    6578616D706C652D706F72742D6661756C74
A2                                     # map(2)
  69                                 # text(9)
    706F72742D6E616D65              # "port-name"
  66                                 # text(6)
    302F342F3231                    # "0/4/21"
  6A                                 # text(10)
    706F72742D6661756C74            # "port-fault"
  6A                                 # text(10)
    4F70656E2070696E2032            # "Open pin 2"

```

4.6. The 'anyxml'

An anyxml representation node is used to serialize an arbitrary CBOR content, i.e., its value can be any CBOR binary object. (The "xml" in the name is a misnomer that only applied to YANG-XML [RFC7950].) An anyxml value MAY contain CBOR data items tagged with one of the tags listed in Section 9.3. The tags listed in Section 9.3 SHALL be supported.

The following example shows a valid CBOR encoded anyxml representation node instance consisting of a CBOR array containing the CBOR simple values 'true', 'null' and 'true'.

Definition example from [RFC7951]:

```

module bar-module {
  ...
  anyxml bar;      # SID 60000
}

```

4.6.1. Using SIDs in keys

CBOR diagnostic notation:

```

{
  60000 : [true, null, true]  / bar (SID 60000) /
}

```

CBOR encoding:


```

A1          # map(1)
  19 EA60  # unsigned(60000)
  83       # array(3)
    F5     # primitive(21)
    F6     # primitive(22)
    F5     # primitive(21)

```

4.6.2. Using names in keys

CBOR diagnostic notation:

```

{
  "bar-module:bar" : [true, null, true]   / bar (SID 60000) /
}

```

CBOR encoding:

```

A1          # map(1)
  6E        # text(14)
    6261722D6D6F64756C653A626172  # "bar-module:bar"
  83       # array(3)
    F5     # primitive(21)
    F6     # primitive(22)
    F5     # primitive(21)

```

5. Encoding of 'yang-data' extension

The yang-data extension [RFC8040] is used to define data structures in YANG that are not intended to be implemented as part of a datastore.

The yang-data extension will specify a container that **MUST** be encoded using the encoding rules of nodes of data trees as defined in Section 4.2.

Just like YANG containers, the yang-data extension can be encoded using either SIDs or names.

Definition example from [I-D.ietf-core-comi] Appendix A:

```
module ietf-coreconf {
  ...

  import ietf-restconf {
    prefix rc;
  }

  rc:yang-data yang-errors {
    container error {
      leaf error-tag {
        type identityref {
          base error-tag;
        }
      }
      leaf error-app-tag {
        type identityref {
          base error-app-tag;
        }
      }
      leaf error-data-node {
        type instance-identifier;
      }
      leaf error-message {
        type string;
      }
    }
  }
}
```

5.1. Using SIDs in keys

The yang-data extensions encoded using SIDs are carried in a CBOR map containing a single item pair. The key of this item is set to the SID assigned to the yang-data extension container; the value is set to the CBOR encoding of this container as defined in Section 4.2.

This example shows a serialization example of the yang-errors yang-data extension as defined in [I-D.ietf-core-com1] using SIDs as defined in Section 3.2.

CBOR diagnostic notation:

```

{
  1024 : {
    4 : 1011,
    1 : 1018,
    2 : 1740,
    3 : "Maximum exceeded"
  }
}

```

/ error (SID 1024) /
/ error-tag (SID 1028) /
/ = invalid-value (SID 1011) /
/ error-app-tag (SID 1025) /
/ = not-in-range (SID 1018) /
/ error-data-node (SID 1026) /
/ = timezone-utc-offset (SID 1740) /
/ error-message (SID 1027) /

CBOR encoding:

```

A1                                     # map(1)
  19 0400                             # unsigned(1024)
  A4                                   # map(4)
    04                               # unsigned(4)
    19 03F3                          # unsigned(1011)
    01                               # unsigned(1)
    19 03FA                          # unsigned(1018)
    02                               # unsigned(2)
    19 06CC                          # unsigned(1740)
    03                               # unsigned(3)
    70                               # text(16)
    4D6178696D756D206578636565646564 # "Maximum exceeded"

```

5.2. Using names in keys

The yang-data extensions encoded using names are carried in a CBOR map containing a single item pair. The key of this item is set to the namespace qualified name of the yang-data extension container; the value is set to the CBOR encoding of this container as defined in Section 4.2.

This example shows a serialization example of the yang-errors yang-data extension as defined in [I-D.ietf-core-comi] using names as defined Section 3.3.

CBOR diagnostic notation:

```

{
  "ietf-coreconf:error" : {
    "error-tag" : "invalid-value",
    "error-app-tag" : "not-in-range",
    "error-data-node" : "timezone-utc-offset",
    "error-message" : "Maximum exceeded"
  }
}

```

CBOR encoding:

```

A1                                     # map(1)
  73                                 # text(19)
    6965746662D636F7265636F6E663A6572726F72  # "ietf-coreconf:error"
  A4                                 # map(4)
    69                                 # text(9)
      6572726F722D746167              # "error-tag"
    6D                                 # text(13)
      696E766616C69642D766616C7565      # "invalid-value"
    6D                                 # text(13)
      6572726F722D6170702D746167        # "error-app-tag"
    6C                                 # text(12)
      6E6F742D696E2D72616E6765          # "not-in-range"
    6F                                 # text(15)
      6572726F722D646174612D6E6F6465      # "error-data-node"
    73                                 # text(19)
      74696D657A6F6E652D7574632D6F66666736574  # "timezone-utc-offset"
    6D                                 # text(13)
      6572726F722D6D657373616765          # "error-message"
    70                                 # text(16)
      4D6178696D756D206578636565646564      # "Maximum exceeded"

```

6. Representing YANG Data Types in CBOR

The CBOR encoding of an instance of a leaf or leaf-list representation node depends on the built-in type of that representation node. The following sub-section defines the CBOR encoding of each built-in type supported by YANG as listed in Section 4.2.4 of [RFC7950]. Each subsection shows an example value assigned to a representation 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 an 'mtu' leaf representation node instance set to 1280 bytes.

Definition example from [RFC8344]:

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

CBOR diagnostic notation: 1280

CBOR encoding: 19 0500

6.2. The integer Types

Leafs of type int8, int16, int32 and int64 MUST be encoded using either CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value.

The following example shows the encoding of a 'timezone-utc-offset' leaf representation node 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 Section 3.4.4 of [RFC8949].

The following example shows the encoding of a 'my-decimal' leaf representation node 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 representation node instance set to "eth0".

Definition example from [RFC8343]:

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

CBOR diagnostic notation: "eth0"

CBOR encoding: 64 65746830

6.5. The 'boolean' Type

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

The following example shows the encoding of an 'enabled' leaf representation node 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, or exceptionally as a tagged text string (see below). Enumeration values are either explicitly assigned using the YANG statement 'value' or automatically assigned based on the algorithm defined in Section 9.6.4.2 of [RFC7950].

The following example shows the encoding of an 'oper-status' leaf representation node 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

Values of 'enumeration' types defined in a 'union' type MUST be encoded using a CBOR text string data item (major type 3) and MUST contain one of the names assigned by 'enum' statements in YANG (see also Section 6.12). The encoding MUST be enclosed by the enumeration CBOR tag as specified in Section 9.3.

Definition example from [RFC7950]:

```
type union {  
  type int32;  
  type enumeration {  
    enum unbounded;  
  }  
}
```

CBOR diagnostic notation: 44("unbounded")

CBOR encoding: D8 2C 69 756E626F756E646564

6.7. The 'bits' Type

Keeping in mind that bit positions are either explicitly assigned using the YANG statement 'position' or automatically assigned based on the algorithm defined in Section 9.7.4.2 of [RFC7950], each element of type bits could be seen as a set of bit positions (or offsets from position 0), that have a value of either 1, which represents the bit being set or 0, which represents that the bit is not set.

Leafs of type bits MUST be encoded either using a CBOR array or byte string (major type 2), or exceptionally as a tagged text string (see below). In case CBOR array representation is used, each element is either a positive integer (major type 0 with value 0 being disallowed) that can be used to calculate the offset of the next byte string, or a byte string (major type 2) that carries the information whether certain bits are set or not. The initial offset value is 0 and each unsigned integer modifies the offset value of the next byte string by the integer value multiplied by 8. For example, if the bit offset is 0 and there is an integer with value 5, the first byte of the byte string that follows will represent bit positions 40 to 47 both ends included. If the byte string has a second byte, it will carry information about bits 48 to 55 and so on. Within each byte, bits are assigned from least to most significant. After the byte string, the offset is modified by the number of bytes in the byte string multiplied by 8. Bytes with no bits set (zero bytes) at the end of the byte string are never generated: If they would occur at the end of the array, the zero bytes are simply omitted; if they occur at the end of a byte string preceding an integer, the zero bytes are removed and the integer adjusted upwards by the number of zero bytes removed. An example follows.

The following example shows the encoding of an 'alarm-state' leaf representation node instance with the 'critical' (position 2), 'warning' (position 8) and 'indeterminate' (position 128) flags set.


```
typedef alarm-state {  
  type bits {  
    bit unknown;  
    bit under-repair;  
    bit critical;  
    bit major;  
    bit minor;  
    bit warning {  
      position 8;  
    }  
    bit indeterminate {  
      position 128;  
    }  
  }  
}  
  
leaf alarm-state {  
  type alarm-state;  
}
```

CBOR diagnostic notation: [h'0401', 14, h'01']

CBOR encoding: 83 42 0401 0E 41 01

In a number of cases the array would only need to have one element -- a byte string with a few bytes inside. For this case, it is REQUIRED to omit the array element and have only the byte array that would have been inside. To illustrate this, let us consider the same example YANG definition, but this time encoding only 'under-repair' and 'critical' flags. The result would be

CBOR diagnostic notation: h'06'

CBOR encoding: 41 06

Elements in the array MUST be either byte strings that do not end in a zero byte, or positive unsigned integers, where byte strings and integers MUST alternate, i.e., adjacent byte strings or adjacent integers are an error. An array with a single byte string MUST instead be encoded as just that byte string. An array with a single positive integer is an error. Note that a recipient can handle trailing zero bytes in the byte strings using the normal rules without any issue, so an implementation MAY silently accept them.

Values of 'bits' types defined in a 'union' type MUST be encoded using a CBOR text string data item (major type 3) and MUST contain a space-separated sequence of names of 'bits' that are set (see also Section 6.12). The encoding MUST be enclosed by the bits CBOR tag as specified in Section 9.3.

The following example shows the encoding of an 'alarm-state' leaf representation node instance defined using a union type with the 'under-repair' and 'critical' flags set.

Definition example:

```
leaf alarm-state-2 {  
  type union {  
    type alarm-state;  
    type bits {  
      bit extra-flag;  
    }  
  }  
}
```

CBOR diagnostic notation: 43("under-repair critical")

CBOR encoding: D8 2B 75 756E6465722D72657061697220637269746963616C

6.8. The 'binary' Type

Leafs of type binary MUST be encoded using a CBOR byte string data item (major type 2).

The following example shows the encoding of an 'aes128-key' leaf representation node 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 representation node referenced by the 'path' YANG statement.

The following example shows the encoding of an 'interface-state-ref' leaf representation node instance set to "eth1".

Definition example from [RFC8343]:

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

CBOR diagnostic notation: "eth1"

CBOR encoding: 64 65746831

6.10. The 'identityref' Type

This specification supports two approaches for encoding identityref: as a YANG Schema Item identifier as defined in Section 3.2, or as a name as defined in Section 6.8 of [RFC7951]. See Section 6.12 for an exceptional case when this representation needs to be tagged.

6.10.1. SIDs as identityref

When representation nodes of type identityref are implemented using SIDs, they MUST be encoded using a CBOR unsigned integer data item (major type 0). (Note that, as they are not used in the position of CBOR map keys, no delta mechanism is employed for SIDs used for identityref.)

The following example shows the encoding of a 'type' leaf representation node instance set to the value 'iana-if-type:ethernetCsmacd' (SID 1880).

Definition example from [RFC7317]:

```
identity interface-type {  
}  
  
identity iana-interface-type {  
    base interface-type;  
}  
  
identity ethernetCsmacd {  
    base iana-interface-type;  
}  
  
leaf type {  
    type identityref {  
        base interface-type;  
    }  
}
```

CBOR diagnostic notation: 1880

CBOR encoding: 19 0758

6.10.2. Name as identityref

Alternatively, an identityref MAY be encoded using a name as defined in Section 3.3. When names are used, identityref MUST be encoded using a CBOR text string data item (major type 3). If the identity is defined in different module than the leaf node containing the identityref data node, the namespace qualified form MUST be used. Otherwise, both the simple and namespace qualified forms are permitted. Names and namespaces are defined in Section 3.3.

The following example shows the encoding of the identity 'iana-if-type:ethernetCsmacd' using its namespace qualified name. This example is described in Section 6.10.1.

CBOR diagnostic notation: "iana-if-type:ethernetCsmacd"

CBOR encoding: 78 1b

69616E612D696662D747970653A65746865726E6574443736D616364

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 an 'is-router' leaf representation node instance when present.

Definition example from [RFC8344]:

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

CBOR diagnostic notation: null

CBOR encoding: F6

6.12. The 'union' Type

Leafs of type union MUST be encoded using the rules associated with one of the types listed. When used in a union, the following YANG datatypes are enclosed by a CBOR tag to avoid confusion between different YANG datatypes encoded using the same CBOR major type.

- * bits
- * enumeration
- * identityref
- * instance-identifier

See Section 9.3 for the assigned value of these CBOR tags.

As mentioned in Section 6.6 and in Section 6.7, 'enumeration' and 'bits' are encoded as a CBOR text string data item (major type 3) when defined within a 'union' type. (This adds considerable complexity, but is necessary because of an idiosyncrasy of the YANG data model for unions; the workaround allows compatibility to be maintained with the encoding of overlapping unions in XML and JSON. See also Section 9.12 of [RFC7950].)

The following example shows the encoding of an 'ip-address' leaf representation node instance when set to "2001:db8:a0b:12f0::1".

Definition example (adapted from [RFC6991]):

```

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

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

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

leaf address {
  type 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 `iIdentifier` as defined in Section 3.2 and one based on names as defined in Section 3.3. See Section 6.12 for an exceptional case when this representation needs to be tagged.

6.13.1. SIDs as instance-identifier

SIDs uniquely identify a schema node. In the case of a single instance schema node, i.e., a schema node defined at the root of a YANG module or submodule or schema nodes defined within a container, the SID is sufficient to identify this instance (representation node). (Note that no delta mechanism is employed for SIDs used for identityref, see Section 6.10.1.)

In the case of a representation node that is an entry of a YANG list, a SID is combined with the list key(s) to identify each instance within the YANG list(s).

Instance identifiers of single instance schema nodes MUST be encoded using a CBOR unsigned integer data item (major type 0) and set to the targeted schema node SID.

Instance identifiers of representation node entries of a YANG list MUST be encoded using a CBOR array data item (major type 4) containing the following entries:

- * The first entry MUST be encoded as a CBOR unsigned integer data item (major type 0) and set to the targeted schema node SID.
- * The following entries MUST contain the value of each key required to identify the instance of the targeted schema node. These keys MUST be ordered as defined in the 'key' YANG statement, starting from the top level list, and followed by each of the subordinate list(s).

Examples within this section assume the definition of a schema node of type 'instance-identifier':

Definition example from [RFC7950]:

```
container system {  
  ...  
  leaf reporting-entity {  
    type instance-identifier;  
  }  
}
```

First example:

The following example shows the encoding of the 'reporting-entity' value referencing data node instance "/system/contact" (SID 1741).

Definition example from [RFC7317]:

```
container system {  
    leaf contact {  
        type string;  
    }  
  
    leaf hostname {  
        type inet:domain-name;  
    }  
}
```

CBOR diagnostic notation: 1741

CBOR encoding: 19 06CD

Second example:

This example aims to show how a representation node entry of a YANG list is identified. It uses a somewhat arbitrarily modified YANG module version from [RFC7317] by adding country to the leafs and keys of authorized-key.

The following example shows the encoding of the 'reporting-entity' value referencing list instance "/system/authentication/user/authorized-key/key-data" (which is assumed to have SID 1734) for username "bob" and authorized-key with name "admin" and country "france".


```
list user {
  key name;

  leaf name {
    type string;
  }

  leaf password {
    type ianach:crypt-hash;
  }

  list authorized-key {
    key "name country";

    leaf country {
      type string;
    }

    leaf name {
      type string;
    }

    leaf algorithm {
      type string;
    }

    leaf key-data {
      type binary;
    }
  }
}
```

CBOR diagnostic notation: [1734, "bob", "admin", "france"]

CBOR encoding:

```
84          # array(4)
 19 06C6    # unsigned(1734)
 63         # text(3)
   626F62   # "bob"
 65         # text(5)
   61646D696E # "admin"
 66         # text(6)
   6672616E6365 # "france"
```

Third example:

The following example shows the encoding of the 'reporting-entity' value referencing the list instance `"/system/authentication/user"` (SID 1730) corresponding to username `"jack"`.

CBOR diagnostic notation: `[1730, "jack"]`

CBOR encoding:

```
82          # array(2)
 19 06C2    # unsigned(1730)
 64         # text(4)
   6A61636B # "jack"
```

6.13.2. Names as instance-identifier

An "instance-identifier" value is encoded as a text string that is analogous to the lexical representation in XML encoding; see Section 9.13.2 of [RFC7950]. However, the encoding of namespaces in instance-identifier values follows the rules stated in Section 3.3, namely:

- * The leftmost (top-level) data node name is always in the namespace qualified form.
- * Any subsequent data node name is in the namespace qualified form if the node is defined in a module other than its parent node, and the simple form is used otherwise. This rule also holds for node names appearing in predicates.

For example,

```
/ietf-interfaces:interfaces/interface[name='eth0']/ietf-ip:ipv4/ip
```

is a valid instance-identifier value because the data nodes "interfaces", "interface", and "name" are defined in the module "ietf-interfaces", whereas "ipv4" and "ip" are defined in "ietf-ip".

The resulting xpath MUST be encoded using a CBOR text string data item (major type 3).

First example:

This example is described in Section 6.13.1.

CBOR diagnostic notation: `"/ietf-system:system/contact"`

CBOR encoding:

```
78 1c 2F696574662D73797374656D3A73797374656D2F636F6E74616374
```

Second example:

This example is described in Section 6.13.1.

CBOR diagnostic notation (the line break is inserted for exposition only):

```
"/ietf-system:system/authentication/user[name='bob']/  
authorized-key[name='admin'][country='france']/key-data"
```

CBOR encoding:

```
78 6B  
2F696574662D73797374656D3A73797374656D2F61757468656E74696361  
74696F6E2F757365725B6E616D653D27626F62275D2F617574686F72697A  
65642D6B65795B6E616D653D2761646D696E275D5B636F756E7472793D27  
6672616E6365275D2F6B65792D64617461
```

Third example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

```
"/ietf-system:system/authentication/user[name='jack']"
```

CBOR encoding:

```
78 34                                     # text(52)  
2F696574662D73797374656D3A73797374656D2F61757468656E74696361  
74696F6E2F757365725B6E616D653D276A61636B275D
```

7. Content-Types

This specification defines the media-type `application/yang-data+cbor`, which can be used without parameters or with the `id` parameter set to either `name` or `sid`.

This media-type represents a YANG-CBOR document containing a representation tree. If the media-type parameter `id` is present, depending on its value, each representation node is identified by its associated namespace qualified name as defined in Section 3.3 (`id=name`), or by its associated YANG SID (represented, e.g., in CBOR map keys as a SID delta or via tag number 47) as defined in Section 3.2 (`id=sid`), respectively. If no `id` parameter is given, both forms may be present.

The format of an application/yang-data+cbor representation is that of a CBOR map, mapping names and/or SIDs (as defined above) into instance values (using the rules defined in Section 4).

It is not foreseen at this point that the valid set of values for the id parameter will extend beyond name, sid, or being unset; if that does happen, any new value is foreseen to be of the form [a-z][a-z0-9]*(-[a-z0-9]+)*.

In summary, this document defines three content-types, which are intended for use by different classes of applications:

- * application/yang-data+cbor; id=sid -- for use by applications that need to be frugal with encoding space and text string processing (e.g., applications running on constrained nodes [RFC7228], or applications with particular performance requirements);
- * application/yang-data+cbor; id=name -- for use by applications that do not want to engage in SID management, and that have ample resources to manage text-string based item identifiers (e.g., applications that directly want to substitute application/yang.data+json with a more efficient representation without any other changes);
- * application/yang-data+cbor -- for use by more complex applications that can benefit from the increased efficiency of SID identifiers but also need to integrate databases of YANG modules before SID mappings are defined for them.

All three content-types are based on the same representation mechanisms, parts of which are simply not used in the first and second case.

How the use of one of these content types is selected in a transfer protocol is outside the scope of this specification. The last paragraph of Section 5.2 of [RFC8040] discusses how to indicate and request the usage of specific content-types in RESTCONF. Similar mechanisms are available in CoAP [RFC7252] using the Content-Format and Accept Options; [I-D.ietf-core-comi] demonstrates specifics on how Content-Format may be used to indicate the id=sid case.

8. Security Considerations

The security considerations of [RFC8949] 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 to 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.

For instance, when the 'id' parameter to the media type is used, it is important to properly reject identifiers of the other type, to avoid scenarios where different implementations interpret a given content in different ways.

When SIDs are in use, the interpretation of encoded data not only relies on having the right YANG modules, but also on having the right SID mapping information. Management and evolution of that mapping information therefore requires the same care as the management and evolution of the YANG modules themselves. The procedures in [I-D.ietf-core-sid] are being defined with this in mind.

9. IANA Considerations

9.1. Media-Types Registry

This document adds the following Media-Type to the "Media Types" registry.

Name	Template	Reference
yang-data+cbor	application/yang-data+cbor	RFC XXXX

Table 2

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

Type name: application
Subtype name: yang-data+cbor
Required parameters: N/A
Optional parameters: id (see Section 7 of RFC XXXX)
Encoding considerations: binary (CBOR)
Security considerations: see Section 8 of RFC XXXX
Published specification: RFC XXXX
Person & email address to contact for further information: CORE WG

mailing list (core@ietf.org), or IETF Applications and Real-Time
Area (art@ietf.org)
Intended usage: COMMON
Restrictions on usage: none
Author/Change controller: IETF

9.2. CoAP Content-Formats Registry

This document adds the following Content-Format to the "CoAP Content-Formats", within the "Constrained RESTful Environments (CoRE) Parameters" registry, where TBD3 comes from the "Expert Review" 0-255 range and TBD1 and TBD2 come from the "IETF Review" 256-9999 range.

Content Type	Content Coding	ID	Reference
application/yang-data+cbor	-	TBD1	RFC XXXX
application/yang-data+cbor; id=name	-	TBD2	RFC XXXX
application/yang-data+cbor; id=sid	-	TBD3	RFC XXXX

Table 3

// RFC Ed.: please replace TBDx with assigned IDs, remove the requested ranges, and remove this note.
// RFC Ed.: please replace RFC XXXX with this RFC number and remove this note.

9.3. CBOR Tags Registry

In the registry "CBOR Tags" [IANA.cbor-tags], as per Section 9.2 of [RFC8949], IANA has allocated the CBOR tags in Table 4 for the YANG datatypes listed.

Tag	Data Item	Semantics	Reference
43	text string	YANG bits datatype; see Section 6.7	RFC XXXX
44	text string	YANG enumeration datatype; see Section 6.6.	RFC XXXX
45	unsigned integer or text string	YANG identityref datatype; see Section 6.10.	RFC XXXX
46	unsigned integer or text string or array	YANG instance-identifier datatype; see Section 6.13.	RFC XXXX
47	unsigned integer	YANG Schema Item identifier (SID); see Section 3.2.	RFC XXXX

Table 4: CBOR tags defined by this specification

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

10. References

10.1. Normative References

- [IANA.cbor-tags]
IANA, "Concise Binary Object Representation (CBOR) Tags",
<<https://www.iana.org/assignments/cbor-tags>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <<https://www.rfc-editor.org/info/rfc5234>>.
- [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>>.
- [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>>.
- [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>>.
- [RFC8610] Birkholz, H., Vigano, C., and C. Bormann, "Concise Data Definition Language (CDDL): A Notational Convention to Express Concise Binary Object Representation (CBOR) and JSON Data Structures", RFC 8610, DOI 10.17487/RFC8610, June 2019, <<https://www.rfc-editor.org/info/rfc8610>>.
- [RFC8791] Bierman, A., Björklund, M., and K. Watsen, "YANG Data Structure Extensions", RFC 8791, DOI 10.17487/RFC8791, June 2020, <<https://www.rfc-editor.org/info/rfc8791>>.
- [RFC8949] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <<https://www.rfc-editor.org/info/rfc8949>>.

10.2. Informative References

- [I-D.ietf-core-comi]
Veillette, M., Stok, P. V. D., Pelov, A., Bierman, A., and I. Petrov, "CoAP Management Interface (CORECONF)", Work in Progress, Internet-Draft, draft-ietf-core-comi-11, 17 January 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-comi-11.txt>>.
- [I-D.ietf-core-sid]
Veillette, M., Pelov, A., Petrov, I., Bormann, C., and M. Richardson, "YANG Schema Item iDentifier (YANG SID)", Work in Progress, Internet-Draft, draft-ietf-core-sid-18, 18 November 2021, <<https://www.ietf.org/archive/id/draft-ietf-core-sid-18.txt>>.

- [RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <<https://www.rfc-editor.org/info/rfc6241>>.
- [RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <<https://www.rfc-editor.org/info/rfc6991>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [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>>.
- [RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <<https://www.rfc-editor.org/info/rfc8343>>.
- [RFC8344] Bjorklund, M., "A YANG Data Model for IP Management", RFC 8344, DOI 10.17487/RFC8344, March 2018, <<https://www.rfc-editor.org/info/rfc8344>>.

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 Jürgen Schönwälder, and from the document shepherd Marco Tiloca. Extensive comments helped us further improve the document in the IESG review process; the authors would like to call out specifically the feedback and guidance by the responsible AD Francesca Palombini and the significant improvements suggested by IESG members Benjamin Kaduk and Rob Wilton.

Authors' Addresses

Michel Veillette (editor)
Trilliant Networks Inc.
610 Rue du Luxembourg
Granby Quebec J2J 2V2
Canada
Email: michel.veillette@trilliantinc.com

Ivaylo Petrov (editor)
Google Switzerland GmbH
Brandschenkestrasse 110
CH-8002 Zurich
Switzerland
Email: ivaylopetrov@google.com

Alexander Pelov
Acklio
1137A avenue des Champs Blancs
35510 Cesson-Sevigne
France
Email: a@ackl.io

Carsten Bormann
Universität Bremen TZI
Postfach 330440
D-28359 Bremen
Germany
Phone: +49-421-218-63921
Email: cabo@tzi.org

Michael Richardson
Sandelman Software Works
Canada
Email: mcr+ietf@sandelman.ca

Network Working Group
Internet-Draft
Intended status: Informational
Expires: March 21, 2019

J. Mattsson
J. Fornehed
G. Selander
F. Palombini
Ericsson
C. Amsuess
Energy Harvesting Solutions
September 17, 2018

Controlling Actuators with CoAP
draft-mattsson-core-coap-actuators-06

Abstract

Being able to trust information from sensors and to securely control actuators are essential in a world of connected and networking things interacting with the physical world. In this memo we show that just using COAP with a security protocol like DTLS, TLS, or OSCORE is not enough. We describe several serious attacks any on-path attacker can do, and discusses tougher requirements and mechanisms to mitigate the attacks. While this document is focused on actuators, some of the attacks apply equally well to sensors.

Status of This Memo

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

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

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

This Internet-Draft will expire on March 21, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

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

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

Table of Contents

1. Introduction	2
1.1. Terminology	4
2. Attacks	4
2.1. The Block Attack	4
2.2. The Request Delay Attack	5
2.3. The Response Delay and Mismatch Attack	8
2.4. The Relay Attack	11
2.5. The Request Fragment Rearrangement Attack	12
2.5.1. Completing an Operation with an Earlier Final Block	13
2.5.2. Injecting a Withheld First Block	14
3. Security Considerations	15
4. IANA Considerations	15
5. References	15
5.1. Normative References	15
5.2. Informative References	16
Acknowledgements	17
Authors' Addresses	17

1. Introduction

Being able to trust information from sensors and to securely control actuators are essential in a world of connected and networking things interacting with the physical world. One protocol used to interact with sensors and actuators is the Constrained Application Protocol (CoAP) [RFC7252]. Any Internet-of-Things (IoT) deployment valuing security and privacy would use a security protocol such as DTLS [RFC6347], TLS [RFC5246], or OSCORE [I-D.ietf-core-object-security] to protect CoAP, where the choice of security protocol depends on the transport protocol and the presence of intermediaries. The use of CoAP over UDP and DTLS is specified in [RFC6347] and the use of CoAP over TCP and TLS is specified in [RFC8323]. OSCORE protects CoAP end-to-end with the use of COSE [RFC8152] and the CoAP Object-Security option [I-D.ietf-core-object-security], and can therefore be used over any transport.

The Constrained Application Protocol (CoAP) [RFC7252] was designed with the assumption that security could be provided on a separate

layer, in particular by using DTLS [RFC6347]. The four properties traditionally provided by security protocols are:

- o Data confidentiality
- o Data origin authentication
- o Data integrity checking
- o Replay protection

In this document we show that protecting CoAP with a security protocol on another layer is not nearly enough to securely control actuators (and in many cases sensors) and that secure operation often demands far more than the four properties traditionally provided by security protocols. We describe several serious attacks any on-path attacker (i.e. not only "trusted intermediaries) can do and discusses tougher requirements and mechanisms to mitigate the attacks. In general, secure operation of actuators also requires the three properties:

- o Data-to-Data binding
- o Data-to-space binding
- o Data-to-time binding

"Data-to-Data binding" is e.g. binding of responses to a request or binding of data fragments to each other. "Data-to-space binding" is the binding of data to an absolute or relative point in space (i.e. a location) and may in the relative case be referred to as proximity. "Data-to-time binding" is the binding of data to an absolute or relative point in time and may in the relative case be referred to as freshness. The two last properties may be bundled together as "Data-to-spacetime binding".

The request delay attack (valid for DTLS, TLS, and OSCORE and described in Section 2.2) lets an attacker control an actuator at a much later time than the client anticipated. The response delay and mismatch attack (valid for DTLS and TLS and described in Section 2.3) lets an attacker respond to a client with a response meant for an older request. The request fragment rearrangement attack (valid for DTLS, TLS, and OSCORE and described in Section 2.5) lets an attacker cause unauthorized operations to be performed on the server, and responses to unauthorized operations to be mistaken for responses to authorized operations.

Mechanisms mitigating some of the attacks discussed in this document can be found in [I-D.ietf-core-echo-request-tag] and [I-D.liu-core-coap-delay-attacks]

1.1. Terminology

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

2. Attacks

Internet-of-Things (IoT) deployments valuing security and privacy, MUST use a security protocol such as DTLS, TLS, or OSCORE to protect CoAP. This is especially true for deployments of actuators where attacks often (but not always) have serious consequences. The attacks described in this section are made under the assumption that CoAP is already protected with a security protocol such as DTLS, TLS, or OSCORE, as an attacker otherwise can easily forge false requests and responses.

2.1. The Block Attack

An on-path attacker can block the delivery of any number of requests or responses. The attack can also be performed by an attacker jamming the lower layer radio protocol. This is true even if a security protocol like DTLS, TLS, or OSCORE is used. Encryption makes selective blocking of messages harder, but not impossible or even infeasible. With DTLS and TLS, proxies have access to the complete CoAP message, and with OSCORE, the CoAP header and several CoAP options are not encrypted. In both security protocols, the IP-addresses, ports, and CoAP message lengths are available to all on-path attackers, which may be enough to determine the server, resource, and command. The block attack is illustrated in Figures 1 and 2.

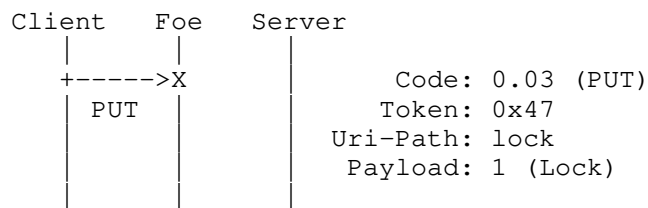


Figure 1: Blocking a request

Where 'X' means the attacker is blocking delivery of the message.

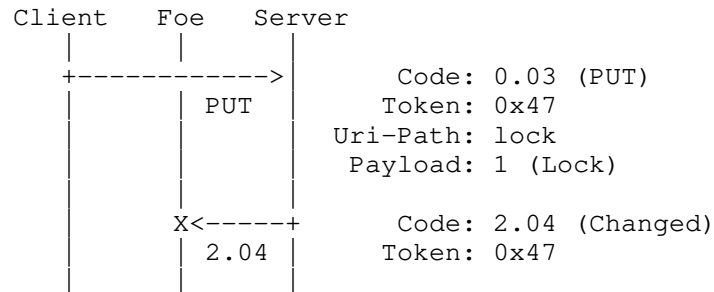


Figure 2: Blocking a response

While blocking requests to, or responses from, a sensor is just a denial of service attack, blocking a request to, or a response from, an actuator results in the client losing information about the server's status. If the actuator e.g. is a lock (door, car, etc.), the attack results in the client not knowing (except by using out-of-band information) whether the lock is unlocked or locked, just like the observer in the famous Schrodinger's cat thought experiment. Due to the nature of the attack, the client cannot distinguish the attack from connectivity problems, offline servers, or unexpected behavior from middle boxes such as NATs and firewalls.

Remedy: Any IoT deployment of actuators where confirmation is important MUST notify the user upon reception of the response, or warn the user when a response is not received.

2.2. The Request Delay Attack

An on-path attacker may not only block packets, but can also delay the delivery of any packet (request or response) by a chosen amount of time. If CoAP is used over a reliable and ordered transport such as TCP with TLS or OSCORE, no messages can be delivered before the delayed message. If CoAP is used over an unreliable and unordered transport such as UDP with DTLS, or OSCORE, other messages can be delivered before the delayed message as long as the delayed packet is delivered inside the replay window. When CoAP is used over UDP, both DTLS and OSCORE allow out-of-order delivery and uses sequence numbers together with a replay window to protect against replay attacks. The replay window has a default length of 64 in DTLS and 32 in OSCORE. The attacker can control the replay window by blocking some or all other packets. By first delaying a request, and then later, after delivery, blocking the response to the request, the client is not made aware of the delayed delivery except by the missing response. The server has in general, no way of knowing that the request was

delayed and will therefore happily process the request. Note that delays can also happen for other reasons than a malicious attacker.

If some wireless low-level protocol is used, the attack can also be performed by the attacker simultaneously recording what the client transmits while at the same time jamming the server. The request delay attack is illustrated in Figure 3.

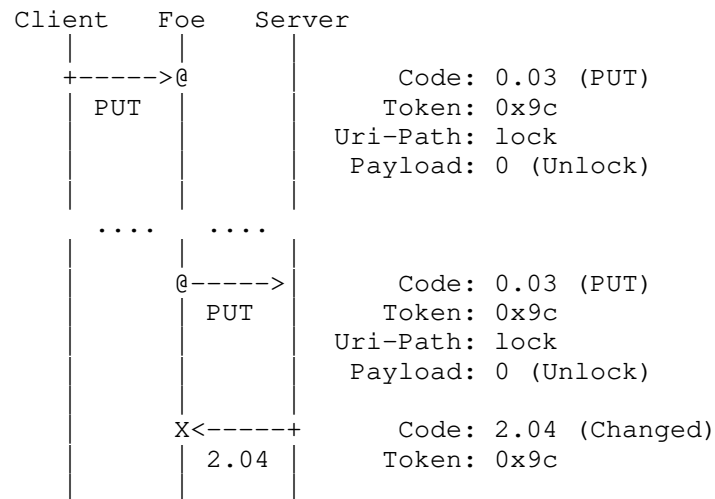


Figure 3: Delaying a request

Where '@' means the attacker is storing and later forwarding the message (@ may alternatively be seen as a wormhole connecting two points in time).

While an attacker delaying a request to a sensor is often not a security problem, an attacker delaying a request to an actuator performing an action is often a serious problem. A request to an actuator (for example a request to unlock a lock) is often only meant to be valid for a short time frame, and if the request does not reach the actuator during this short timeframe, the request should not be fulfilled. In the unlock example, if the client does not get any response and does not physically see the lock opening, the user is likely to walk away, calling the locksmith (or the IT-support).

If a non-zero replay window is used (the default when CoAP is used over UDP), the attacker can let the client interact with the actuator before delivering the delayed request to the server (illustrated in Figure 4). In the lock example, the attacker may store the first "unlock" request for later use. The client will likely resend the request with the same token. If DTLS is used, the resent packet will

have a different sequence number and the attacker can forward it. If OSCORE is used, resent packets will have the same sequence number and the attacker must block them all until the client sends a new message with a new sequence number (not shown in Figure 4). After a while when the client has locked the door again, the attacker can deliver the delayed "unlock" message to the door, a very serious attack.

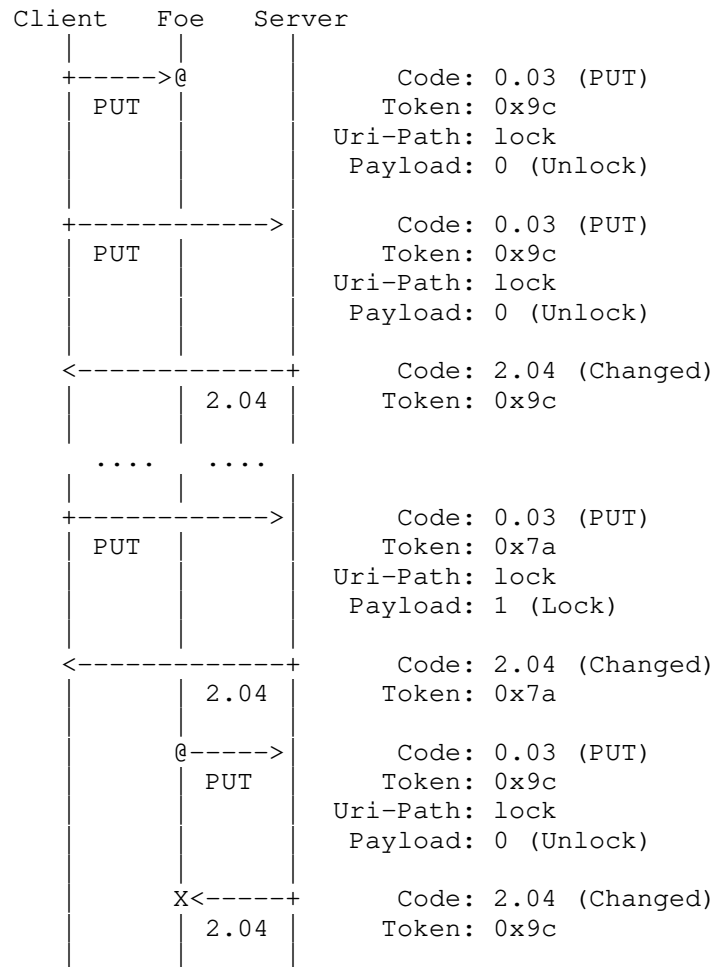


Figure 4: Delaying request with reordering

While the second attack (Figure 4) can be mitigated by using a replay window of length zero, the first attack (Figure 3) cannot. A solution must enable the server to verify that the request was received within a certain time frame after it was sent or enable the server to securely determine an absolute point in time when the

request is to be executed. This can be accomplished with either a challenge-response pattern, by exchanging timestamps between client and server, or by only allowing requests a short period after client authentication.

Requiring a fresh client authentication (such as a new TLS/DTLS handshake or an EDHOC key exchange [I-D.selander-ace-cose-ecdhe]) mitigates the problem, but requires larger messages and more processing than a dedicated solution. Security solutions based on exchanging timestamps require exactly synchronized time between client and server, and this may be hard to control with complications such as time zones and daylight saving. Wall clock time SHOULD NOT be used as it is not monotonic, may reveal that the endpoints will accept expired certificates, or reveal the endpoint's location. Use of non-monotonic clocks is not secure as the server will accept requests if the clock is moved backward and reject requests if the clock is moved forward. Even if the clocks are synchronized at one point in time, they may easily get out-of-sync and an attacker may even be able to affect the client or the server time in various ways such as setting up a fake NTP server, broadcasting false time signals to radio controlled clocks, or expose one of them to a strong gravity field. As soon as client falsely believes it is time synchronized with the server, delay attacks are possible. A challenge response mechanism where the server does not need to synchronize its time with the client is easier to analyze but require more roundtrips. The challenges, responses, and timestamps may be sent in a CoAP option or in the CoAP payload.

Remedy: The mechanisms specified in [I-D.ietf-core-echo-request-tag] or [I-D.liu-core-coap-delay-attacks] SHALL be used for controlling actuators unless another application specific challenge-response or timestamp mechanism is used.

2.3. The Response Delay and Mismatch Attack

The following attack can be performed if CoAP is protected by a security protocol where the response is not bound to the request in any way except by the CoAP token. This would include most general security protocols, such as DTLS, TLS, and IPsec, but not OSCORE. CoAP [RFC7252] uses a client generated token that the server echoes to match responses to request, but does not give any guidelines for the use of token with DTLS and TLS, except that the tokens currently "in use" SHOULD (not SHALL) be unique. The attacker performs the attack by delaying delivery of a response until the client sends a request with the same token, the response will be accepted by the client as a valid response to the later request. If CoAP is used over a reliable and ordered transport such as TCP with TLS, no messages can be delivered before the delayed message. If CoAP is

used over an unreliable and unordered transport such as UDP with DTLS, other messages can be delivered before the delayed message as long as the delayed packet is delivered inside the replay window. Note that mismatches can also happen for other reasons than a malicious attacker, e.g. delayed delivery or a server sending notifications to an uninterested client.

The attack can be performed by an attacker on the wire, or an attacker simultaneously recording what the server transmits while at the same time jamming the client. The response delay and mismatch attack is illustrated in Figure 5.

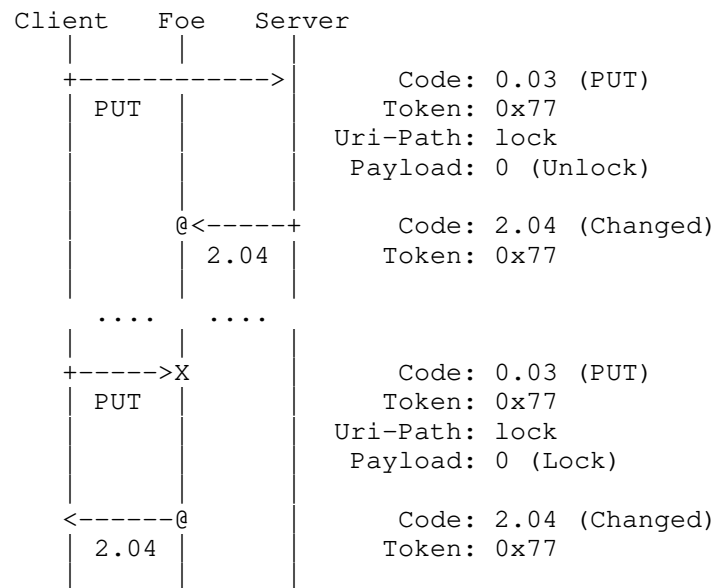


Figure 5: Delaying and mismatching response to PUT

If we once again take a lock as an example, the security consequences may be severe as the client receives a response message likely to be interpreted as confirmation of a locked door, while the received response message is in fact confirming an earlier unlock of the door. As the client is likely to leave the (believed to be locked) door unattended, the attacker may enter the home, enterprise, or car protected by the lock.

The same attack may be performed on sensors, also this with serious consequences. As illustrated in Figure 6, an attacker may convince the client that the lock is locked, when it in fact is not. The "Unlock" request may be also be sent by another client authorized to control the lock.

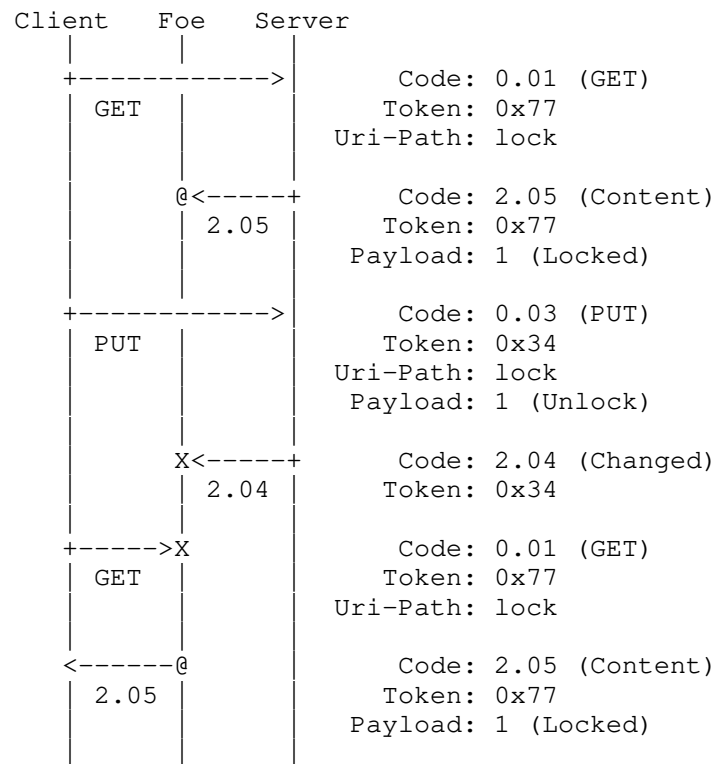


Figure 6: Delaying and mismatching response to GET

As illustrated in Figure 7, an attacker may even mix responses from different resources as long as the two resources share the same (D)TLS connection on some part of the path towards the client. This can happen if the resources are located behind a common gateway, or are served by the same CoAP proxy. An on-path attacker (not necessarily a (D)TLS endpoint such as a proxy) may e.g. deceive a client that the living room is on fire by responding with an earlier delayed response from the oven (temperatures in degree Celsius).

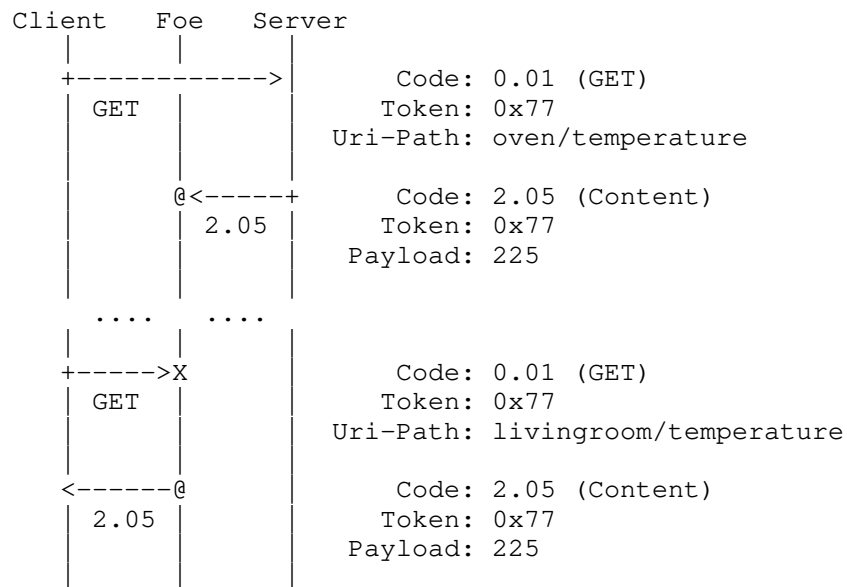


Figure 7: Delaying and mismatching response from other resource

Remedy: If CoAP is protected with a security protocol not providing bindings between requests and responses (e.g. DTLS and TLS) the client MUST NOT reuse any tokens until the traffic keys have been replaced. The easiest way to accomplish this is to implement the Token as a counter, this approach SHOULD be followed.

2.4. The Relay Attack

Yet another type of attack can be performed in deployments where actuator actions are triggered automatically based on proximity and without any user interaction, e.g. a car (the client) constantly polling for the car key (the server) and unlocking both doors and engine as soon as the car key responds. An attacker (or pair of attackers) may simply relay the CoAP messages out-of-band, using for examples some other radio technology. By doing this, the actuator (i.e. the car) believes that the client is close by and performs actions based on that false assumption. The attack is illustrated in Figure 8. In this example the car is using an application specific challenge-response mechanism transferred as CoAP payloads.

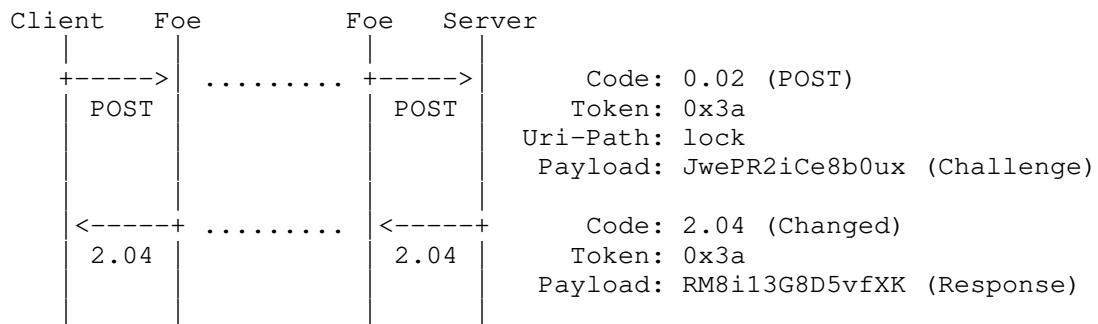


Figure 8: Relay attack (the client is the actuator)

The consequences may be severe, and in the case of a car, lead to the attacker unlocking and driving away with the car, an attack that unfortunately is happening in practice.

Remedy: Getting a response over a short-range radio MUST NOT be taken as proof of proximity and therefore MUST NOT be used to take actions based on such proximity. Any automatically triggered mechanisms relying on proximity MUST use other stronger mechanisms to guarantee proximity. Mechanisms that MAY be used are: measuring the round-trip time and calculate the maximum possible distance based on the speed of light, or using radio with an extremely short range like NFC (centimeters instead of meters) that cannot be relayed through e.g. clothes. Another option is to including geographical coordinates (from e.g. GPS) in the messages and calculate proximity based on these, but in this case the location measurements MUST be very precise and the system MUST make sure that an attacker cannot influence the location estimation, something that is very hard in practice.

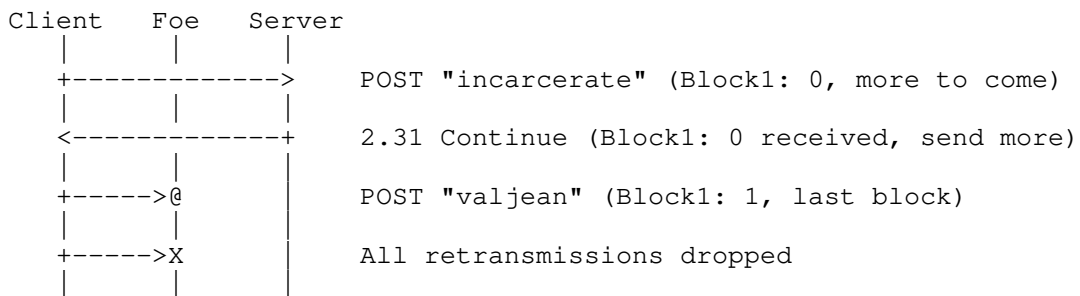
2.5. The Request Fragment Rearrangement Attack

These attack scenarios show that the Request Delay and Block Attacks can be used against blockwise transfers to cause unauthorized operations to be performed on the server, and responses to unauthorized operations to be mistaken for responses to authorized operations. The combination of these attacks is described as a separate attack because it makes the Request Delay Attack relevant to systems that are otherwise not time-dependent, which means that they could disregard the Request Delay Attack.

This attack works even if the individual request/response pairs are encrypted, authenticated and protected against the Response Delay and Mismatch Attack, provided the attacker is on the network path and can correctly guess which operations the respective packages belong to.

2.5.1. Completing an Operation with an Earlier Final Block

In this scenario (illustrated in Figure 9), blocks from two operations on a POST-accepting resource are combined to make the server execute an action that was not intended by the authorized client. This works only if the client attempts a second operation after the first operation failed (due to what the attacker made appear like a network outage) within the replay window. The client does not receive a confirmation on the second operation either, but, by the time the client acts on it, the server has already executed the unauthorized action.



(Client: Odd, but let's go on and promote Javert)

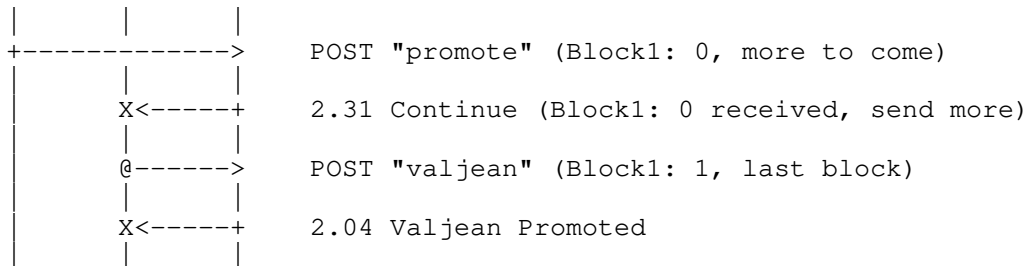


Figure 9: Completing an operation with an earlier final block

Remedy: If a client starts new blockwise operations on a security context that has lost packages, it needs to label the fragments in such a way that the server will not mix them up.

A mechanism to that effect is described as Request-Tag [I-D.ietf-core-echo-request-tag]. Had it been in place in the example and used for body integrity protection, the client would have set the Request-Tag option in the "promote" request. Depending on the server's capabilities and setup, either of four outcomes could have occurred:

1. The server could have processed the reinjected POST "valjean" as belonging to the original "incarcerate" block; that's the expected case when the server can handle simultaneous block transfers.
2. The server could respond 5.03 Service Unavailable, including a Max-Age option indicating how long it prefers not to take any requests that force it to overwrite the state kept for the "incarcerate" request.
3. The server could decide to drop the state kept for the "incarcerate" request's state, and process the "promote" request. The reinjected POST "valjean" will then fail with 4.08 Request Entity incomplete, indicating that the server does not have the start of the operation any more.

2.5.2. Injecting a Withheld First Block

If the first block of a request is withheld by the attacker for later use, it can be used to have the server process a different request body than intended by the client. Unlike in the previous scenario, it will return a response based on that body to the client.

Again, a first operation (that would go like "Homeless stole apples. What shall we do with him?" - "Set him free.") is aborted by the proxy, and a part of that operation is later used in a different operation to prime the server for responding leniently to another operation that would originally have been "Hitman killed someone. What shall we do with him?" - "Hang him.". The attack is illustrated in Figure 10.

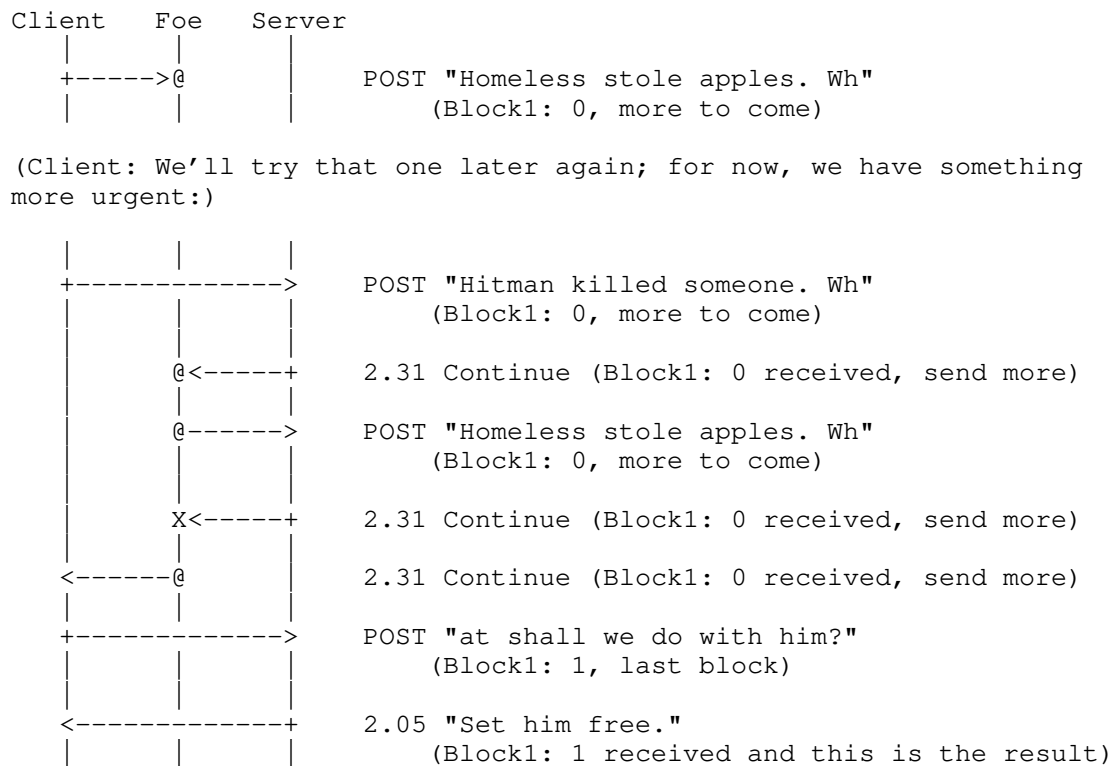


Figure 10: Injecting a withheld first block

3. Security Considerations

The whole document can be seen as security considerations for CoAP.

4. IANA Considerations

This document has no actions for IANA.

5. References

5.1. Normative References

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

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <<https://www.rfc-editor.org/info/rfc7252>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.

5.2. Informative References

- [I-D.ietf-core-echo-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Echo and Request-Tag", draft-ietf-core-echo-request-tag-02 (work in progress), June 2018.
- [I-D.ietf-core-object-security]
Selander, G., Mattsson, J., Palombini, F., and L. Seitz, "Object Security for Constrained RESTful Environments (OSCORE)", draft-ietf-core-object-security-15 (work in progress), August 2018.
- [I-D.liu-core-coap-delay-attacks]
Liu, Y. and J. Zhu, "Mitigating delay attacks on Constrained Application Protocol", draft-liu-core-coap-delay-attacks-01 (work in progress), October 2017.
- [I-D.selander-ace-cose-ecdhe]
Selander, G., Mattsson, J., and F. Palombini, "Ephemeral Diffie-Hellman Over COSE (EDHOC)", draft-selander-ace-cose-ecdhe-09 (work in progress), July 2018.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<https://www.rfc-editor.org/info/rfc6347>>.
- [RFC8152] Schaad, J., "CBOR Object Signing and Encryption (COSE)", RFC 8152, DOI 10.17487/RFC8152, July 2017, <<https://www.rfc-editor.org/info/rfc8152>>.

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

Acknowledgements

The authors would like to thank Carsten Bormann, Klaus Hartke, Ari Keraenen, Matthias Kovatsch, Sandeep Kumar, and Andras Mehes for their valuable comments and feedback.

Authors' Addresses

John Mattsson
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com

John Fornehed
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: john.fornehed@ericsson.com

Goeran Selander
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
SE-164 80 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Christian Amsuess
Energy Harvesting Solutions

Email: c.amsuess@energyharvesting.at

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 15, 2018

J. Mattsson
Ericsson AB
November 11, 2017

Message Size Overhead of CoAP Security Protocols
draft-mattsson-core-security-overhead-02

Abstract

This document analyzes and compares per-packet message size overheads when using different security protocols to secure CoAP. The analyzed security protocols are DTLS 1.2, DTLS 1.3, TLS 1.2, TLS 1.3, and OSCORE. DTLS and TLS are analyzed with and without compression. DTLS are analyzed with two different alternatives for header compression as well as with and without Connection ID.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 15, 2018.

Copyright Notice

Copyright (c) 2017 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. Overhead of Security Protocols	3
2.1. DTLS	3
2.1.1. DTLS 1.2	3
2.1.2. DTLS 1.2 with 6LoWPAN-GHC	3
2.1.3. DTLS 1.2 with raza-6lo-compressed-dtls	4
2.1.4. DTLS 1.3	5
2.1.5. DTLS 1.3 with 6LoWPAN-GHC	5
2.1.6. DTLS 1.3 with raza-6lo-compressed-dtls	6
2.2. DTLS with Connection ID	7
2.2.1. DTLS 1.2 with Connection ID	7
2.2.2. DTLS 1.2 with Connection ID and 6LoWPAN-GHC	7
2.2.3. DTLS 1.3 with Connection ID	8
2.2.4. DTLS 1.3 with Connection ID and 6LoWPAN-GHC	9
2.3. TLS	10
2.3.1. TLS 1.2	10
2.3.2. TLS 1.2 with 6LoWPAN-GHC	10
2.3.3. TLS 1.3	11
2.3.4. TLS 1.3 with 6LoWPAN-GHC	11
2.4. OSCORE	12
3. Overhead with Different Parameters	14
4. Summary	15
5. Security Considerations	16
6. Informative References	16
Acknowledgments	18
Author's Address	18

1. Introduction

This document analyzes and compares per-packet message size overheads when using different security protocols to secure CoAP over UDP [RFC7252] and TCP [I-D.ietf-core-coap-tcp-tls]. The analyzed security protocols are DTLS 1.2 [RFC6347], DTLS 1.3 [I-D.rescorla-tls-dtls13], TLS 1.2 [RFC5246], TLS 1.3 [I-D.ietf-tls-tls13], and OSCORE [I-D.ietf-core-object-security]. The DTLS and TLS record layers are analyzed with and without compression. DTLS are analyzed with two different alternatives ([RFC7400] and [raza-6lo-compressed-dtls]) for header compression as well as with and without Connection ID [I-D.rescorla-tls-dtls-connection-id].

2. Overhead of Security Protocols

To enable comparison, all the overhead calculations in this section use AES-CCM with a tag length of 8 bytes, a plaintext of 6 bytes, and the sequence number '05'. This follows the example in [RFC7400], Figure 16.

2.1. DTLS

2.1.1. DTLS 1.2

This section analyzes the overhead of DTLS 1.2 [RFC6347]. The nonce follow the strict profiling given in [RFC7925]. This example is taken directly from [RFC7400], Figure 16. .

DTLS 1.2 Record Layer (35 bytes, 29 bytes overhead):

```
17 fe fd 00 01 00 00 00 00 00 05 00 16 00 01 00
00 00 00 00 05 ae a0 15 56 67 92 4d ff 8a 24 e4
cb 35 b9
```

Content type:

17

Version:

fe fd

Epoch:

00 01

Sequence number:

00 00 00 00 00 05

Length:

00 16

Nonce:

00 01 00 00 00 00 00 05

Ciphertext:

ae a0 15 56 67 92

ICV:

4d ff 8a 24 e4 cb 35 b9

DTLS 1.2 gives 29 bytes overhead.

2.1.2. DTLS 1.2 with 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.2 [RFC6347] when compressed with [RFC7400]. The compression was done with [OlegHahn-ghc].

Note that the compressed overhead is dependent on the parameters epoch, sequence number, and length. The following is only an example.

Note that the sequence number '01' used in [RFC7400], Figure 15 gives an exceptionally small overhead that is not representative.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.2 Record Layer (22 bytes, 16 bytes overhead):

```
b0 c3 03 05 00 16 f2 0e ae a0 15 56 67 92 4d ff
8a 24 e4 cb 35 b9
```

Compressed DTLS 1.2 Record Layer Header and Nonce:

```
b0 c3 03 05 00 16 f2 0e
```

Ciphertext:

```
ae a0 15 56 67 92
```

ICV:

```
4d ff 8a 24 e4 cb 35 b9
```

When compressed with 6LoWPAN-GHC, DTLS 1.2 with the above parameters (epoch, sequence number, length) gives 16 bytes overhead.

2.1.3. DTLS 1.2 with raza-6lo-compressed-dtls

This section analyzes the overhead of DTLS 1.2 [RFC6347] when compressed with [raza-6lo-compressed-dtls].

Note that the compressed overhead is dependent on the parameters epoch and sequence number. The following is only an example.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with raza-6lo-compressed-dtls.

Compressed DTLS 1.2 Record Layer (19 bytes, 13 bytes overhead):

```
90 17 01 00 05 ae a0 15 56 67 92 4d ff 8a 24 e4
cb 35 b9
```

NHC

```
90
```

Compressed DTLS 1.2 Record Layer Header and Nonce:

```
17 01 00 05
```

Ciphertext:

```
ae a0 15 56 67 92
```

ICV:

```
4d ff 8a 24 e4 cb 35 b9
```

When compressed with raza-6lo-compressed-dtls, DTLS 1.2 with the above parameters (epoch, sequence number) gives 13 bytes overhead.

2.1.4. DTLS 1.3

This section analyzes the overhead of DTLS 1.3 [I-D.rescorla-tls-dtls13]. The only change compared to DTLS 1.2 is that the DTLS 1.3 record layer does not have an explicit nonce.

DTLS 1.3 Record Layer (27 bytes, 21 bytes overhead):
17 fe fd 00 01 00 00 00 00 00 05 00 0e ae a0 15
56 67 92 4d ff 8a 24 e4 cb 35 b9

Content type:
17
Version:
fe fd
Epoch:
00 01
Sequence number:
00 00 00 00 00 05
Length:
00 0e
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 gives 21 bytes overhead.

2.1.5. DTLS 1.3 with 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.3 [I-D.rescorla-tls-dtls13] when compressed with [RFC7400] [OlegHahm-ghc].

Note that the overhead is dependent on the parameters epoch, sequence number, and length. The following is only an example.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.3 Record Layer (20 bytes, 14 bytes overhead):
b0 c3 11 05 00 0e ae a0 15 56 67 92 4d ff 8a 24
e4 cb 35 b9

Compressed DTLS 1.3 Record Layer Header and Nonce:
b0 c3 11 05 00 0e
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.3 with the above parameters (epoch, sequence number, length) gives 14 bytes overhead.

2.1.6. DTLS 1.3 with raza-6lo-compressed-dtls

This section analyzes the overhead of DTLS 1.3 [I-D.rescorla-tls-dtls13] when compressed with [raza-6lo-compressed-dtls].

Note that the compressed overhead is dependent on the parameters epoch and sequence number. The following is only an example.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with raza-6lo-compressed-dtls.

Compressed DTLS 1.3 Record Layer (19 bytes, 13 bytes overhead):
90 17 01 00 05 ae a0 15 56 67 92 4d ff 8a 24 e4
cb 35 b9

NHC
90
Compressed DTLS 1.3 Record Layer Header and Nonce:
17 01 00 05
c3 03 05 00 16 f2 0e
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with raza-6lo-compressed-dtls, DTLS 1.3 with the above parameters (epoch, sequence number) gives 13 bytes overhead.

2.2. DTLS with Connection ID

This section analyzes the overhead of DTLS with Connection ID [I-D.rescorla-tls-dtls-connection-id]. The overhead calculations in this section uses Connection ID = '42'. DTLS with a Connection ID = '' (the empty string) is equal to DTLS without Connection ID.

2.2.1. DTLS 1.2 with Connection ID

This section analyzes the overhead of DTLS 1.2 [RFC6347] with Connection ID [I-D.rescorla-tls-dtls-connection-id].

Note that the overhead is dependent on the parameter Connection ID. The following is only an example.

DTLS 1.2 Record Layer (35 bytes, 29 bytes overhead):

```
17 fe fd 00 01 00 00 00 00 00 05 42 00 16 00 01
00 00 00 00 00 05 ae a0 15 56 67 92 4d ff 8a 24
e4 cb 35 b9
```

Content type:

17

Version:

fe fd

Epoch:

00 01

Sequence number:

00 00 00 00 00 05

Connection ID:

42

Length:

00 16

Nonce:

00 01 00 00 00 00 00 05

Ciphertext:

ae a0 15 56 67 92

ICV:

4d ff 8a 24 e4 cb 35 b9

DTLS 1.2 with Connection ID gives 30 bytes overhead.

2.2.2. DTLS 1.2 with Connection ID and 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.2 [RFC6347] with Connection ID [I-D.rescorla-tls-dtls-connection-id] when compressed with [RFC7400] [OlegHahm-ghc].

Note that the compressed overhead is dependent on the parameters epoch, sequence number, Connection ID, and length. The following is only an example.

Note that the sequence number '01' used in [RFC7400], Figure 15 gives an exceptionally small overhead that is not representative.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.2 Record Layer (23 bytes, 17 bytes overhead):

```
b0 c3 04 05 42 00 16 f2 0e ae a0 15 56 67 92 4d
ff 8a 24 e4 cb 35 b9
```

Compressed DTLS 1.2 Record Layer Header and Nonce:

```
b0 c3 04 05 42 00 16 f2 0e
```

Ciphertext:

```
ae a0 15 56 67 92
```

ICV:

```
4d ff 8a 24 e4 cb 35 b9
```

When compressed with 6LoWPAN-GHC, DTLS 1.2 with the above parameters (epoch, sequence number, Connection ID, length) gives 17 bytes overhead.

2.2.3. DTLS 1.3 with Connection ID

This section analyzes the overhead of DTLS 1.3 [I-D.rescorla-tls-dtls13] with Connection ID [I-D.rescorla-tls-dtls-connection-id].

Note that the overhead is dependent on the parameter Connection ID. The following is only an example.

DTLS 1.3 Record Layer (28 bytes, 22 bytes overhead):
17 fe fd 00 01 00 00 00 00 00 05 42 00 0e ae a0
15 56 67 92 4d ff 8a 24 e4 cb 35 b9

Content type:
17
Version:
fe fd
Epoch:
00 01
Sequence number:
00 00 00 00 00 05
Connection ID:
42
Length:
00 0e
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

DTLS 1.3 gives 22 bytes overhead.

2.2.4. DTLS 1.3 with Connection ID and 6LoWPAN-GHC

This section analyzes the overhead of DTLS 1.3
[I-D.rescorla-tls-dtls13] with Connection ID
[I-D.rescorla-tls-dtls-connection-id] when compressed with [RFC7400]
[OlegHahm-ghc].

Note that the overhead is dependent on the parameters epoch, sequence number, Connection ID, and length. The following is only an example.

Note that this header compression is not available when DTLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed DTLS 1.3 Record Layer (21 bytes, 15 bytes overhead):
b0 c3 12 05 42 00 0e ae a0 15 56 67 92 4d ff 8a
24 e4 cb 35 b9

Compressed DTLS 1.3 Record Layer Header and Nonce:
b0 c3 12 05 42 00 0e
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, DTLS 1.3 with the above parameters (epoch, sequence number, Connection ID, length) gives 15 bytes overhead.

2.3. TLS

2.3.1. TLS 1.2

This section analyzes the overhead of TLS 1.2 [RFC5246]. The changes compared to DTLS 1.2 is that the TLS 1.2 record layer does not have epoch and sequence number, and that the version is different.

TLS 1.2 Record Layer (27 bytes, 21 bytes overhead):
17 03 03 00 16 00 00 00 00 00 00 05 ae a0 15
56 67 92 4d ff 8a 24 e4 cb 35 b9

Content type:

17

Version:

03 03

Length:

00 16

Nonce:

00 00 00 00 00 00 00 05

Ciphertext:

ae a0 15 56 67 92

ICV:

4d ff 8a 24 e4 cb 35 b9

TLS 1.2 gives 21 bytes overhead.

2.3.2. TLS 1.2 with 6LoWPAN-GHC

This section analyzes the overhead of TLS 1.2 [RFC5246] when compressed with [RFC7400] [OlegHahm-ghc].

Note that the overhead is dependent on the parameters epoch, sequence number, and length. The following is only an example.

Note that this header compression is not available when TLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed TLS 1.2 Record Layer (23 bytes, 17 bytes overhead):
05 17 03 03 00 16 85 0f 05 ae a0 15 56 67 92 4d
ff 8a 24 e4 cb 35 b9

Compressed TLS 1.2 Record Layer Header and Nonce:
05 17 03 03 00 16 85 0f 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, TLS 1.2 with the above parameters (epoch, sequence number, length) gives 17 bytes overhead.

2.3.3. TLS 1.3

This section analyzes the overhead of TLS 1.3 [I-D.ietf-tls-tls13]. The change compared to TLS 1.2 is that the TLS 1.3 record layer uses a different version.

TLS 1.3 Record Layer (27 bytes, 21 bytes overhead):
17 03 01 00 16 00 00 00 00 00 00 00 05 ae a0 15
56 67 92 4d ff 8a 24 e4 cb 35 b9

Content type:
17
Version:
03 01
Length:
00 16
Nonce:
00 00 00 00 00 00 00 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

TLS 1.3 gives 21 bytes overhead.

2.3.4. TLS 1.3 with 6LoWPAN-GHC

This section analyzes the overhead of TLS 1.3 [I-D.ietf-tls-tls13] when compressed with [RFC7400] [OlegHahm-ghc].

Note that the overhead is dependent on the parameters epoch, sequence number, and length. The following is only an example.

Note that this header compression is not available when TLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC.

Compressed TLS 1.3 Record Layer (23 bytes, 17 bytes overhead):
02 17 03 c3 01 16 85 0f 05 ae a0 15 56 67 92 4d
ff 8a 24 e4 cb 35 b9

Compressed TLS 1.3 Record Layer Header and Nonce:
02 17 03 c3 01 16 85 0f 05
Ciphertext:
ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

When compressed with 6LoWPAN-GHC, TLS 1.3 with the above parameters (epoch, sequence number, length) gives 17 bytes overhead.

2.4. OSCORE

This section analyzes the overhead of OSCORE [I-D.ietf-core-object-security].

Note that the overhead is dependent on the included CoAP Option numbers as well as the length of the OSCORE parameters Sender ID and sequence number.

Note that Sender ID = '' (empty string) can only be used by one client per server.

The examples below assume that the original messages does not have payload (note that this does not affect the overhead).

The below calculation Option Delta = '9', Sender ID = '' (empty string), and Sequence Number = '05', and is only an example.

OSCORE Request (19 bytes, 13 bytes overhead):
92 09 05
ff ec ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

CoAP Option Delta and Length
92
Option Value (flag byte and sequence number):
09 05
Payload Marker
ff
Ciphertext (including encrypted code):
ec ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

The below calculation Option Delta = '9', Sender ID = '42', and Sequence Number = '05', and is only an example.

OSCORE Request (20 bytes, 14 bytes overhead):
93 09 05 42
ff ec ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

CoAP Option Delta and Length
93
Option Value (flag byte, sequence number, and Sender ID):
09 05 42
Payload Marker
ff
Ciphertext (including encrypted code):
ec ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

The below calculation uses Option Delta = '9' and is only an example.

OSCORE Response (17 bytes, 11 bytes overhead):
90
ff ec ae a0 15 56 67 92 4d ff 8a 24 e4 cb 35 b9

CoAP Delta and Option Length:
90
Option Value
-
Payload Marker
ff
Ciphertext (including encrypted code):
ec ae a0 15 56 67 92
ICV:
4d ff 8a 24 e4 cb 35 b9

OSCORE with the above parameters gives 13-14 bytes overhead for requests and 11 bytes overhead for responses.

Unlike DTLS and TLS, OSCORE has much smaller overhead for responses than requests.

3. Overhead with Different Parameters

The DTLS overhead is dependent on the parameter Connection ID. The following overheads apply for all Connection IDs with the same length.

The compression overhead (GHC) is dependent on the parameters epoch, sequence number, Connection ID, and length. The following overheads should be representative for sequence numbers and Connection IDs with the same length.

The compression overhead (raza-6lo-compressed-dtls) is dependent on the length of the parameters epoch and sequence number. The following overheads apply for all sequence numbers with the same length.

The OSCORE overhead is dependent on the included CoAP Option numbers as well as the length of the OSCORE parameters Sender ID and sequence number. The following overheads apply for all sequence numbers and Sender IDs with the same length.

Sequence Number	'05'	'1005'	'100005'
DTLS 1.2	29	29	29
DTLS 1.3	21	21	21
TLS 1.2	21	21	21
TLS 1.3	21	21	21
DTLS 1.2 (Raza)	13	13	14
DTLS 1.3 (Raza)	13	13	14
DTLS 1.2 (GHC)	16	16	17
DTLS 1.3 (GHC)	14	14	15
TLS 1.2 (GHC)	17	18	19
TLS 1.3 (GHC)	17	18	19
OSCORE Request	13	14	15
OSCORE Response	11	11	11

Figure 1: Overhead as a function of sequence number
(Connection/Sender ID = '')

Connection/Sender ID	' '	'42'	'4002'
DTLS 1.2	29	30	31
DTLS 1.3	21	22	23
DTLS 1.2 (GHC)	16	17	18
DTLS 1.3 (GHC)	14	15	16
OSCORE Request	13	14	15
OSCORE Response	11	11	11

Figure 2: Overhead as a function of Connection/Sender ID
(Sequence Number = '05')

4. Summary

DTLS 1.2 has quite a large overhead as it uses an explicit sequence number and an explicit nonce. DTLS 1.3, TLS 1.2, and TLS 1.3 have significantly less (but not small) overhead.

Both DTLS compression methods provides very good compression. raza-6lo-compressed-dtls achieves slightly better compression but requires state. GHC is stateless but provides slightly worse compression. As DTLS 1.3 uses the same version number as DTLS 1.2, both GHC and raza-6lo-compressed-dtls works well also for DTLS 1.3.

The Generic Header Compression (6LoWPAN-GHC) can in addition to DTLS 1.2 handle DTLS 1.3, DTLS with Connection ID, TLS 1.2, and TLS 1.3. The Generic Header Compression (6LoWPAN-GHC) works very well for Connection ID and the overhead seems to increase exactly with the length of the Connection ID (which is optimal). The compression of TLS is not as good as the compression of DTLS (as the static dictionary is more or less a DTLS record layer). Similar compression levels as for DTLS could be achieved also for TLS, but this would require different static dictionaries for each version of TLS (as TLS 1.2 and TLS 1.3 uses different version numbers). GHC works as good for DTLS 1.3 as for DTLS 1.2 as the version number is the same.

raza-6lo-compressed-dtls is not able to handle DTLS with Connection ID or TLS, all extensions requires an updated mechanism.

The header compression is not available when (D)TLS is exchanged over transports that do not use 6LoWPAN together with 6LoWPAN-GHC or raza-6lo-compressed-dtls.

OSCORE has much lower overhead than DTLS and TLS. The overhead of OSCORE is smaller than DTLS over 6LoWPAN with compression, and this small overhead is achieved even on deployments without 6LoWPAN or 6LoWPAN without DTLS compression. OSCORE is lightweight because it makes use of some excellent features in CoAP, CBOR, and COSE.

5. Security Considerations

This document is purely informational.

6. Informative References

[I-D.ietf-core-coap-tcp-tls]

Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-10 (work in progress), October 2017.

[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-06 (work in progress), October 2017.

[I-D.ietf-tls-tls13]

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", draft-ietf-tls-tls13-21 (work in progress), July 2017.

- [I-D.rescorla-tls-dtls-connection-id]
Rescorla, E. and H. Tschofenig, "The Datagram Transport Layer Security (DTLS) Connection Identifier", draft-rescorla-tls-dtls-connection-id-01 (work in progress), October 2017.
- [I-D.rescorla-tls-dtls13]
Rescorla, E., Tschofenig, H., and N. Modadugu, "The Datagram Transport Layer Security (DTLS) Protocol Version 1.3", draft-rescorla-tls-dtls13-01 (work in progress), March 2017.
- [OlegHahm-ghc]
Hahm, O., "Generic Header Compression", July 2016, <<https://github.com/OlegHahm/ghc>>.
- [raza-6lo-compressed-dtls]
Raza, S., Shafagh, H., and O. Dupont, "Compression of Record and Handshake Headers for Constrained Environments", March 2017, <<http://shahidraza.info/draft-raza-6lo-compressed.txt>>.
- [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>>.
- [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>>.
- [RFC7400] Bormann, C., "6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 7400, DOI 10.17487/RFC7400, November 2014, <<https://www.rfc-editor.org/info/rfc7400>>.
- [RFC7925] Tschofenig, H., Ed. and T. Fossati, "Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things", RFC 7925, DOI 10.17487/RFC7925, July 2016, <<https://www.rfc-editor.org/info/rfc7925>>.

Acknowledgments

The authors want to thank Ari Keraenen, Francesca Palombini, and Goeran Selander for reviewing previous versions of the draft.

All 6LoWPAN-GHC compression was done with [OlegHahm-ghc].

Author's Address

John Mattsson
Ericsson AB
Faeroegatan 6
Kista SE-164 80 Stockholm
Sweden

Email: john.mattsson@ericsson.com

ANIMA WG
Internet-Draft
Intended status: Informational
Expires: May 4, 2017

M. Pritikin
P. Kampanakis
Cisco Systems
October 31, 2016

BRSKI over CoAP
draft-pritikin-coap-bootstrap-01

Abstract

This document provides an initial discussion of Bootstrapping of Remote Secure key infrastructures (BRSKI) when the device being bootstrapped speaks CoAP. The HTTPS REST methods leveraged by BRSKI are mapped to CoAP methods. Fragmentation management of large messages during EST certificate enrollment is addressed.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 4, 2017.

Copyright Notice

Copyright (c) 2016 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	2
3. Scope of solution	3
4. DTLS	3
5. Message Bindings	4
5.1. cacerts	5
5.2. enroll / reenroll	6
5.3. csrattr	7
5.4. requestaudittoken, requestauditlog	7
6. Data Fragmentation	7
6.1. Fragmented response example with Block2	8
6.2. Fragmented request example with Block1	11
6.3. Fragmented request/response example with Block1, Size1 and Block2	11
7. Proxying	11
8. CoAP Parameters	11
9. Security Considerations	11
10. Update Tracking	11
11. Normative References	11
Authors' Addresses	12

1. Introduction

Many IoT and other devices are expected to use CoAP over UDP extensively. Bootstrapping these devices without requiring a full TCP stack is an often raised requirement for [I-D.ietf-anima-bootstrapping-keyinfra]. BRSKI provides REST methods over TLS that can be functional in a UDP setting with the following necessary additions:

DTLS: Because the CoAP use of DTLS includes support for large handshake messages there is little to describe here. BRSKI and EST [RFC7030] are expanded to include DTLS.

REST: The mapping of BRSKI and EST messages to CoAP REST calls is described.

Fragmentation: Use of block chaining to support fragmentation of large BRSKI and EST messages is described.

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].

3. Scope of solution

The definition of BRSKI over DTLS and CoAP is not intended to expand the scope of BRSKI to highly constrained devices. (ref: [RFC7228]). Instead it is intended to ensure that bootstrapping works for less constrained devices that choose to limit their communications stack to UDP/CoAP.

The BRSKI document details extensions to EST as well as making section 5.7 requirements on EST flows. This document's references to BRSKI are intended to include all BRSKI extensions and all existing EST messages. This document could replace BRSKI -03 section 5.7.5. [[TODO: making this section 5.8 might make the most sense.]]

Support for Observe CoAP options (<https://tools.ietf.org/html/rfc7641>) in Blocks with BRSKI is not supported in the current BRSKI/EST message flows and is thus out-of-scope for this discussion. Observe options could be used by the server to notify clients about a change in the cacerts or csr attributes (resources) and might be an area of future work.

4. DTLS

COAP was designed to avoid fragmentation. DTLS is used to secure COAP messages. When using DTLS, even though it can be avoided by using pre-shared keys or ECC ciphersuites, sometimes fragmentation will be needed. During the DTLS handshake, if fragmentation is necessary, "DTLS provides a mechanism for fragmenting a handshake message over a number of records, each of which can be transmitted separately, thus avoiding IP fragmentation" [RFC6347].

Within BRSKI and EST when "TLS" is referred to, it is understood that CoAP security is provided using DTLS instead. No other changes are necessary (all provisional modes etc are the same as for TLS).

In a constrained CoAP environment, endpoints can't afford to establish a DTLS connection for every EST transaction. Authenticating and negotiating DTLS keys requires resources on low-end endpoints and consumes valuable bandwidth. The DTLS connection SHOULD remain open for persistent EST connections. For example, an EST cacerts request that is followed by an simpleenroll request can use the same authenticated DTLS connection. Given that after a successful enrollment, it is more likely that a new EST transaction will take place after a significant amount of time, the DTLS connections SHOULD only be kept alive for EST messages that are relatively close to each other.

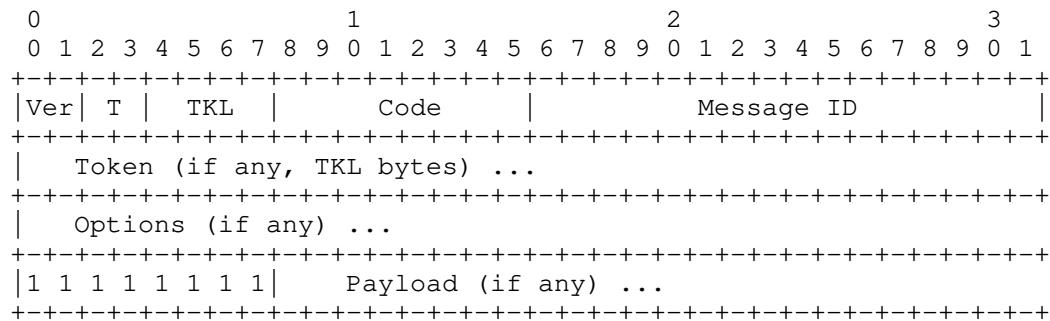
5. Message Bindings

This section describes BRSKI to CoAP message mappings.

CoAP defines confirmed (CON), acknowledgements (ACK), reset (RST) and non-confirmed (NON) message types. For confirmable messages, the responses are CoAP ACKs or RSTs. All /cacerts, /simpleenroll, /simplereenroll, /csrattrs, /fullcmc and /serverkeygen EST messages expect a response, so they are all CoAP CON messages.

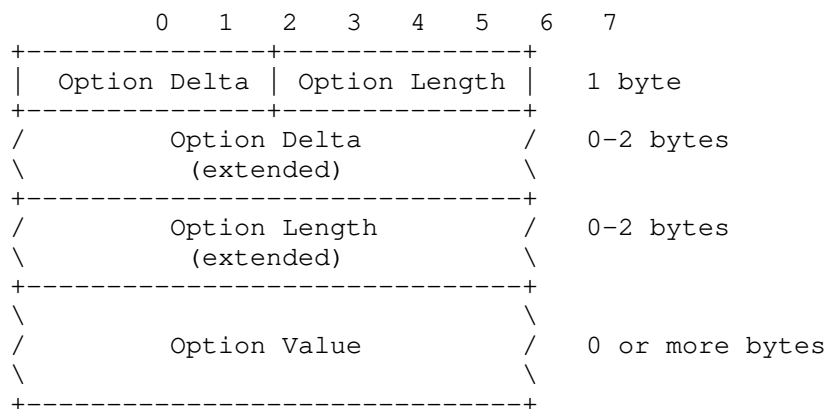
The HTTP responses in BRSKI are translated to CoAP Response Codes as explained in [RFC7252] section 5.3.1

A CoAP message has the following fields ([I-D.ietf-core-block]):



Then Ver, TKL, Token, Message ID are not affected in BRSKI. Their use is the same as in CoAP.

The options that can be used in a CoAP header have the following format (from [I-D.ietf-core-block] Figure 8):



Options are used to convey Uri-Host, Uri-Path, Uri-Port, Content-Format and more in CoAP. For BRSKI, CoAP Options are used to communicate the HTTP fields used in the BRSKI REST messages.

BRSKI URLs are HTTPS based (`https://`), in CoAP these will be assumed to be transformed to `coaps` (`coaps://`)

Some examples of how an BRSKI message would be translated in CoAP follow. `[[TODO: This section to be expanded to ensure it covers all BRSKI edge conditions.]]`

5.1. cacerts

First let's see how a `get cacerts` message in EST would be in CoAP. The HTTPS `cacerts` message can be

```
GET /.well-known/est/cacerts HTTP/1.1
User-Agent: curl/7.22.0 (i686-pc-linux-gnu) libcurl/7.22.0
           OpenSSL/1.0.1 zlib/1.2.3.4 libidn/1.23 librtmp/2.3
Host: 192.0.2.1:8085
Accept: */*
```

The corresponding CoAP fields would be:

```
Ver = 1
T = 0 (CON)
Code = 0x01 (0.01 is GET)
Options
Option1 (Uri-Host)
  Option Delta = 0x3
  Option Length = 0x9
  Option Value = 192.0.2.1
Option2 (Uri-Port)
  Option Delta = 0xA
  Option Length = 0x4
  Option Value = 8085
Option3 (Uri-Path)
  Option Delta = 0xD
  Option Length = 0xD
  Extended Option Delta = 0x08
  Extended Option Length = 0x14
  Option Value = /.well-known/est/cacerts HTTP/1.1
Payload = [Empty]
```

Now let's say we have a 200 OK response with a cert in EST:

```

HTTP/1.1 200 OK
Status: 200 OK
Content-Type: application/pkcs7-mime
Content-Transfer-Encoding: base64 (TODO: Verify if we need a new
                                option registry for Encoding?)
Content-Length: 4246 (TODO: this example overflows and would
                        need fragmentation. Choose a better example.
                        Regardless we might need an CoAP option for
                        the content-length ie the CoAP payload?)

```

```

MIIMOQYJKoZIhvcNAQcCoIIMKjCCDCYCAQExADALBgkqhkiG9w0BBwGgggMMIIC
+zCCAeOgAwIBAgIJAjPy3nUZO3qcMA0GCSqGSIb3DQEBBQUAMBSxGTAXBgNVBAMT
...

```

The corresponding CoAP fields would be:

```

Ver = 1
T = 2 (ACK)
Code = 0x21 (TODO: Maybe we need to create a 0x200 response code.)
Options
  Option1 (Content-Format)
    Option Delta = 0xC
    Option Length = 0xD
    Extended Option Length = 0x09
    Option Value = <number for application/pkcs7-mime>
                  (TODO: We need a new CoAP IANA registered value
                    application/pkcs7-mime; smime-type=certs-only,
                    application/csrattrs, application/pkcs10,
                    application/pkcs8,
                    application/pkcs12 )
Payload = MIIMOQYJKoZIhvcNAQcCoIIMKjCCDCYCAQExA \
          DALBgkqhkiG9w0BBwGgggMMIIC...

```

5.2. enroll / reenroll

[[TODO: username/password authentication can be described here but is not a primary focus for BRSKI. It is important for generic EST exchanges but would an endpoint device with sufficient user interface to allow username/password input from an end user be required to use CoAP instead of a full HTTPS exchange?]]

[[TODO: We might need a new Option for the Retry-After response message. We might need a new Option for the WWW-Authenticate response.]]

5.3. csrattr

5.4. requestaudittoken, requestauditlog

6. Data Fragmentation

Even though ECC certificates are small in size, they can vary greatly based on signature algorithms, key sizes, and OID fields used. Even with ECC certs, BRSKI CoAP messages carrying such certificates can still exceed sizes in MTU of 1280 for IPv6 or 60-80 bytes for 6LoWPAN [RFC4919]) (section 2 of [I-D.ietf-core-block]). For 256-bit curves, common ECDSA cert sizes are 500-1000 bytes which could fluctuate further based on the algorithms, OIDs, SANs and cert fields. For 384-bit curves, ECDSA certs increase in size and can sometimes reach 1.5KB. Additionally, there are times when the EST cacert response from the server can include multiple certs that amount large payloads.

CoAP RFC section 4.6 describes the possible payload sizes: "if nothing is known about the size of the headers, good upper bounds are 1152 bytes for the message size and 1024 bytes for the payload size". Also "If IPv4 support on unusual networks is a consideration, implementations may want to limit themselves to more conservative IPv4 datagram sizes such as 576 bytes; per [RFC0791], the absolute minimum value of the IP MTU for IPv4 is as low as 68 bytes, which would leave only 40 bytes minus security overhead for a UDP payload". Thus, after the DTLS connection is established, fragmentation will sometimes be needed for the CoAP messages which involve certificate enrollment and management. A fragmentation solution for BRSKI and EST CoAP message is required.

The [I-D.ietf-core-block] document describes how fragmentation can be done by using a pair of Block options added to the CoAP flow. Block1 options are used by the client PUT and POST requests. A Block1 in a client request requires a Block1 option in the responses. A Block2 comes from a server response that will also need Block2 from the client to acknowledge the block and get the rest of blocks from the server. So, Block1 is used when a request (POST for example) is done in BRSKI over CoAP with a payload that needs fragmentation. Then the server responds with Block1 option to acknowledge the fragment-blocks. Block2 is used when a BRSKI server response is big and needs fragmentation. The Block2 acknowledgements are requests with the same options as the initial request and a Block2 option. "To influence the block size used in a response, the requester MAY also use the Block2 Option on the initial request, giving the desired size, a block number of zero and an M bit of zero". As explained in see section 2.8 of [I-D.ietf-core-block]), blockwise transfers SHOULD

be used in Confirmable COAP messages to avoid the exacerbation of lost blocks.

In a scenario with a big BRSKI POST request we might have Block1 options from client to server and Block2 from server to client. In this case the Block1 blocks get completed and then the Block2 comes the other direction.

The BLOCK draft also defines Size1 and Size2 options. These are used to convey the size of the resources in the requests or responses. The Size1 response MAY be parsed by the client as an size indication of the Block2 resource in the server response or by the server as a request for a size estimate by the client. Similarly, Size2 option defined in BLOCK should be parsed by the server as an indication of the size of the resource carried in Block1 options and by the client as a maximum size expected in the 4.13 (Request Entity Too Large) response to a request.

6.1. Fragmented response example with Block2

An example of a server cacerts response that exceeds the MTU is:

An example of a server cacerts response that exceeds the MTU is
HTTP/1.1 200 OK

Status: 200 OK

Content-Type: application/pkcs7-mime; smime-type=certs-only

Content-Transfer-Encoding: base64

Content-Length: 1122

```
MIIDOAYJKoZIhvcNAQcCoIIDKTCCAYUCAQExADALBgkqhkiG9w0BBwGgggMLMIID
BzCCAe+gAwIBAgIBFTANBgkqhkiG9w0BAQUFADAbMRkwFwYDVQQDExBlc3RFeGFT
cGxlQ0EgTndOMB4XDTEzMDUwOTIzMTU1M1oXDTE0MDUwOTIzMTU1M1owHzEdMBsG
A1UEAxMUZGVtb3N0ZXA0IDEzNjgxNDEzNTIwggEiMA0GCSqGSIb3DQEBAQUAA4IB
DwAwggEKAoIBAQC1Np+kdz+Nj8XpEp9kaumWxDZ3eFYJpQKz9ddD5e5OzUeCm103
ZIXQIxc0eVtMCatnRr3dnZRCAXGjwbqoB3eKt29/XSQffVv+odbyw0WdkQOIbntC
Qry8YdcBZ+8LjI/N7M2krmjmoSLmLwU2V4aNKf0YMLR5Krmah3Ik31jmYCSvwTnv
6mx6pr2pTJ82JavhTEIIt/fAYq1RYhkM1CXoBL+yhEoDanN7TzC94skfS3VV+f53
J9SkUxTYcy1Rw0k3VXfxWwy+cSKEPRE17I6k0YeKtDEVAgBIEYM/L1S69RXTLuji
rwnqSRjOquzkAkD31BE961KZCxeYGrhxaR4PAgMBAAGjUjBQMA4GA1UdDwEB/wQE
AwIESDAdBgNVHQ4EFgQU/qDdB6ii6icQ8wGMXvy1jfe4xtUwHwYDVR0jBBgwFoAU
scRp5lujBKfYl6OLO7+5arIyQjwwDQYJKoZIhvcNAQEFBQADggEBACmxg1hvL6+7
a+lFTARoxainBx5gxdZ9omSb0L+qL+4PDvg/+KHZKsDnMCrcU6M4YP5n0EDKmGa6
4lY8fbET4tt7juJg6ixb95/760Th0vuctwkGr6+D6ETTfgyHnrbhX31AhnB+0Ja7
olgv4CWxh1I8aRaTXdpOHORvN0SMXdcr1Cys2vrt0l+LjR2a3kaJJO6eQ5leOdZF
QlZfOPhaLWen0e2BLNJI0vsC2Fa+2LMcnfC38XfGALa5A8e7fNHXWZBjXZLBCza3
rEs9Mlh2CjA/ocSC/WxmMvd+Eqnt/FpggRy+F8IZSRvBarUctGE1lgDmu6AFUxce
R4P0rT2xz8ChADEA
```

Block options in CoAP messages can contain fields, SZX, M and NUM which are not affected by BRSKI.

Let's assume that the cacerts message will need to be broken up to 3 messages. The first Block2 will be:

```
Ver = 1
T = 2 (ACK)
Code = 0x21 (2.01 success message.
      TODO: Do we need to create a 0x200 respond code.)
Options
  Option1 (Content-Format)
    Option Delta = 0xC
    Option Length = 0xD
    Extended Option Length = 0x09
    Option Value = <number for application/pkcs7-mime;
                  smime-type=certs-only>
                  (TODO: We need a new CoAP IANA registered value
                   application/pkcs7-mime, application/csrattrs,
                   application/pkcs10, application/pkcs8,
                   application/pkcs12 )
  Option2 (Block2)
    Option Delta = 0xD
    Option Length = 0x1
    Extended Option Delta = 0x16
    Option Value = 0x0D
Payload = MIIMOQYJKoZIhvcNAQcCoIIMKjCCDCYC \
        AQExADALBgkqhkiG9w0BBwGgggwMMIIC... (512 bytes)
```

The second Block2:

```
Ver = 1
T = 2 (means ACK)
Code = 0x21
Options
  Option1 (Content-Format)
    Option Delta = 0xC
    Option Length = 0xD
    Extended Option Length = 0x09
    Option Value = <number for application/pkcs7-mime;
                  smime-type=certs-only>
                  (TODO: We need a new CoAP IANA registered value
                    application/pkcs7-mime, application/csrattrs,
                    application/pkcs10, application/pkcs8,
                    application/pkcs12 )
  Option2 (Block2)
    Option Delta = 0xD
    Option Length = 0x1
    Extended Option Delta = 0x16
    Option Value = 0x1D
Payload = ... (512 bytes)
```

The third and final Block2:

```
Ver = 1
T = 2 (means ACK)
Code = 0x21
Options
  Option1 (Content-Format)
    Option Delta = 0xC
    Option Length = 0xD
    Extended Option Length = 0x09
    Option Value = <number for application/pkcs7-mime;
                  smime-type=certs-only>
                  (TODO: We need a new CoAP IANA registered value
                    application/pkcs7-mime, application/csrattrs,
                    application/pkcs10, application/pkcs8,
                    application/pkcs12 )
  Option2 (Block2)
    Option Delta = 0xD
    Option Length = 0x1
    Extended Option Delta = 0x16
    Option Value = 0x25
Payload = ...
```


6.2. Fragmented request example with Block1

6.3. Fragmented request/response example with Block1, Size1 and Block2

7. Proxying

[[TODO: This section to be populated. It will address how proxying can take place by an entity that resides at the edge of the CoAP network, such as the Registrar, and can reach the BRSKI server residing in a traditional "TCP setting".]]

8. CoAP Parameters

[[TODO: This section to be populated. It will address transmission parameters for BRSKI described in sections 4.7 and 4.8 of the CoAP draft. BRSKI does not impose any unique parameters that affect the CoAP parameters in Table 2 and 3 in the CoAP draft but the ones in CoAP could be affecting BRSKI. For example the processing delay of CAs could be less than 2s, but in this case they should send a CoAP ACK every 2s while processing.]]

9. Security Considerations

[[TODO: This section to be populated. This document describes an existing protocol moved to CoAP and there should not be additional security concerns added beyond the protocol's or CoAP's specifics security considerations.]]

10. Update Tracking

-01:

Added more binding usecases and Block examples subsections to be expanded on later. Made various text improvements and clarifications.

Added two clarifying sentences in the relevant sections to address Brian C.'s comments and explain that message fragmentation can be avoided to a degree in DTLS and that fragments of block transfers should be confirmed to prevent exacerbated fragment loss in constrained networks.

11. Normative References

- [I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., and S.
Bjarnason, "Bootstrapping Remote Secure Key
Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-
keyinfra-03 (work in progress), June 2016.
- [I-D.ietf-core-block]
Bormann, D. and Z. Shelby, "Block-wise transfers in CoAP",
draft-ietf-core-block-20 (work in progress), April 2016.
- [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>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed.,
"Enrollment over Secure Transport", RFC 7030,
DOI 10.17487/RFC7030, October 2013,
<<http://www.rfc-editor.org/info/rfc7030>>.
- [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for
Constrained-Node Networks", RFC 7228,
DOI 10.17487/RFC7228, May 2014,
<<http://www.rfc-editor.org/info/rfc7228>>.
- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained
Application Protocol (CoAP)", RFC 7252,
DOI 10.17487/RFC7252, June 2014,
<<http://www.rfc-editor.org/info/rfc7252>>.

Authors' Addresses

Max Pritikin
Cisco Systems

Email: pritikin@cisco.com

Panos Kampanakis
Cisco Systems

Email: pkampana@cisco.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.

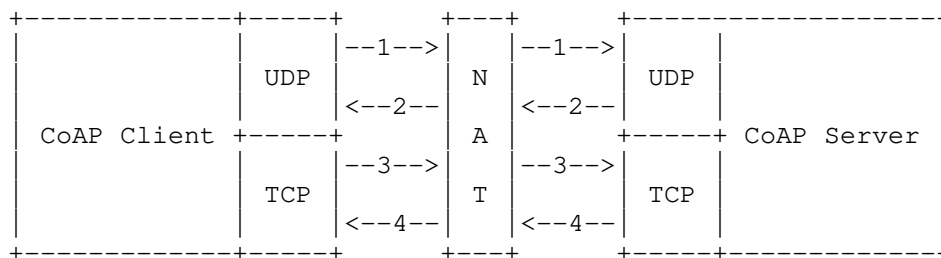


Figure 1: CoAP Client initially accesses CoAP Server over UDP and then switching to TCP

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.

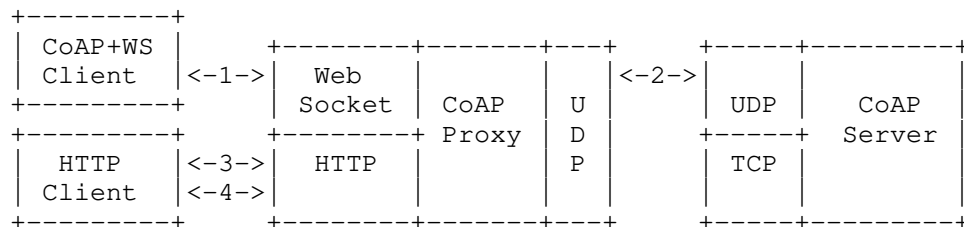


Figure 2: Proxying and returning a resource's alternate cached representations to multiple clients

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.

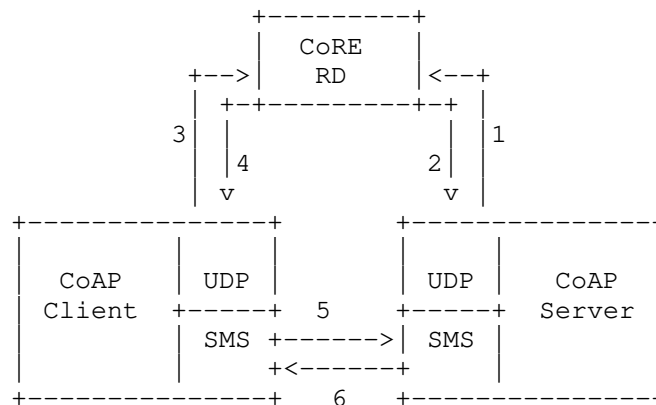


Figure 3: CoAP client interacting with RD to discover a server's SMS-based endpoint

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.

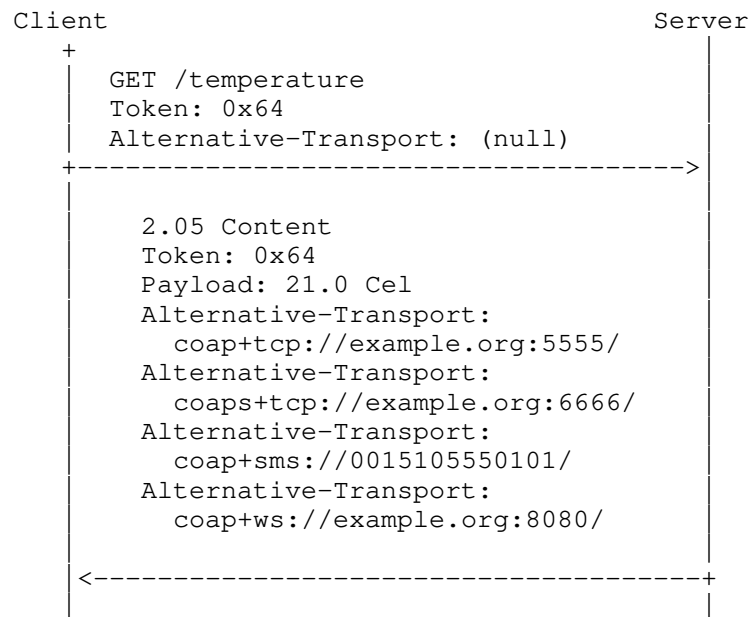


Figure 5: Requesting all available alternative transports on the server, and their locations

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.

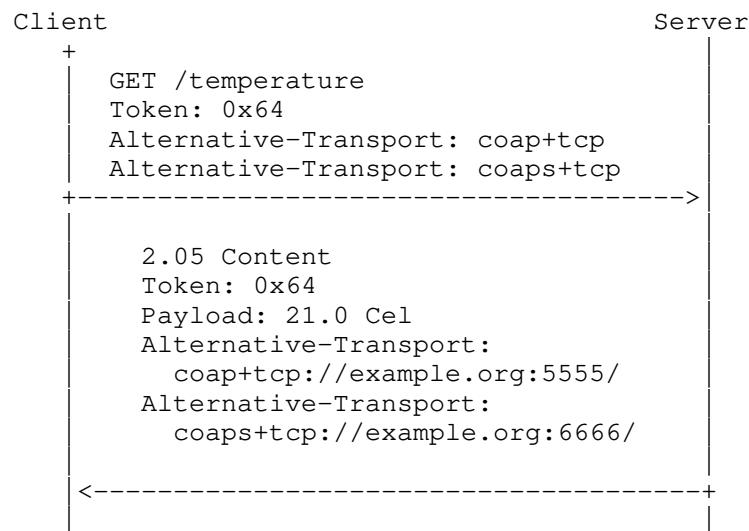


Figure 6: Requesting TCP-based alternative transports on the server, and their locations

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.

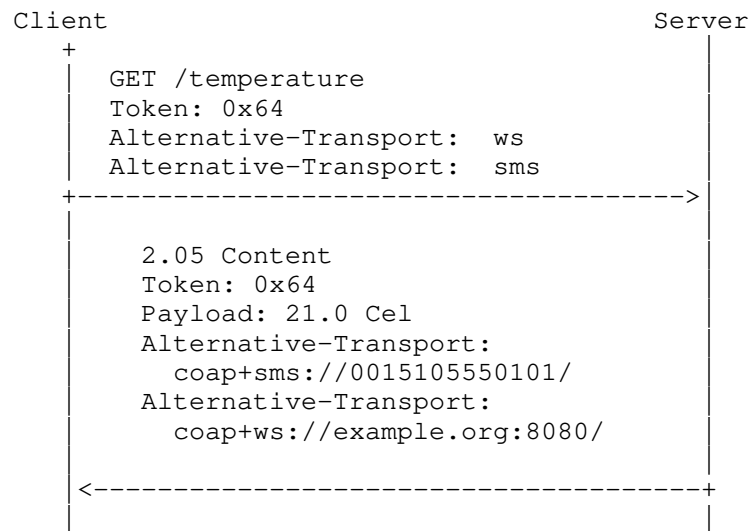


Figure 7: Requesting WebSocket- and SMS-based alternative transports on the server, and their locations

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:5
683/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

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 4, 2017

D. Thaler
Microsoft
October 31, 2016

COAP Redirects
draft-thaler-core-redirect-01

Abstract

This document allows a Constrained Application Protocol (CoAP) server to redirect a client to a new URI. The primary use case is to allow a client using multicast CoAP discovery to learn a COAPS endpoint of the server, without the server revealing privacy-sensitive information. This improves security and privacy in environments with untrusted clients.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 4, 2017.

Copyright Notice

Copyright (c) 2016 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
1.1. Example	3
2. Alternatives Considered	4
2.1. Just use normal multicast discovery	4
2.2. Just use a resource directory	4
2.3. Use Alternative-Address	5
3. Redirects	5
3.1. Option Definitions	5
3.1.1. Location-Scheme and Location-Authority	5
3.2. Response Codes	6
3.2.1. 3.01 Moved Permanently	6
4. IANA Considerations	6
5. Security Considerations	7
6. References	7
6.1. Normative References	7
6.2. Informative References	7
Author's Address	8

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a specialized web transfer protocol for use with constrained nodes and constrained networks. When COAP nodes can appear on a network that allows untrusted clients, security and privacy issues can arise, as discussed in Section 11 of [RFC7252].

This document focuses on a solution for a specific use case: preventing privacy-sensitive information from being passed to untrusted clients, especially as part of resource discovery. The resource discovery phase is important because DTLS is not used with multicast COAP.

The specific relevant threats are:

- o Correlation across location: If a COAP server can move between multiple networks in which an attacker has a presence, the attacker can potentially correlate responses from the COAP server across the two locations and determine that the same entity is moving between those two locations. This can even be used to identify individuals, such as when the COAP server is in a wearable device.

- o Correlation across time: If a COAP server is available periodically in the same location over a long time, an attacker in that location can potentially correlate responses over time and determine that it is the same entity, even though the IP address and layer-2 address may be different. This can even be used to identify individuals, such as when the COAP server is in a wearable device.
- o Fingerprinting: Device-specific vulnerability exploitation can be most easily accomplished if an attacker can easily narrow down what software the server runs. Information returned via multicast service discovery can facilitate such fingerprinting.

For more discussion of these threats, see Section 5.2 of [RFC6973], Section 3 of [RFC7721], and [I-D.winfaa-intarea-broadcast-consider].

To mitigate these threats, this document defines the ability for a server to redirect a client to another URI. Specifically, the expected use is that in response to an unsecured COAP request, a privacy-sensitive server could be configured to simply respond by redirecting the client to a COAPS endpoint, thus allowing the client to discover a unicast endpoint, but not to discover any privacy-sensitive information without establishing a secured unicast connection.

By comparison, HTTP (Section 6.4.2 of [RFC7231]) redirects with 301 (Moved Permanently) and a Location header containing the new URI. COAP, on the other hand, defines Location-Path and Location-Query COAP options [RFC7252] for those components of the URI, but did not define options for the other URI components. [ListDiscussion] explains:

While early drafts of CoAP did have some forms of redirection, we found that the use cases most people had in mind did not call for redirects. The main reason is that in a CoRE world, URIs are usually found through a discovery process, and these URIs can be made to point to the right place right away.

The use case motivating this document, however, is specifically for redirects as part of the discovery process itself.

1.1. Example

Existing clients conforming to the OIC 1.1 Core spec [OIC1.1Core] sections 10 and 11.3.5 do discovery by sending a multicast CoAP GET for "/oic/res". Existing servers will respond with links to a set of resources, but that information might be privacy-sensitive in some cases. For example, it might contain sufficient a unique identifier

of the server, or information sufficient for an attacker to determine what version of what software it runs. (A sample response can be found in section 10.2 of [OIC1.1Core].) Hence a privacy-sensitive server needs a way to be discovered by trusted clients without revealing privacy-sensitive information to untrusted observers. A redirect allows a client to send the same request, thus not increasing the amount of multicast traffic on the network.

For example, consider a network with a privacy-sensitive server, and a legacy server. A client wants to efficiently discover both servers. The client can send a single multicast GET for "/oic/res", and the legacy server would send a unicast response with the requested data, whereas the privacy-sensitive server would respond with a unicast redirect to "coaps://<ipaddr>:<port>/oic/res". The client can then generate a unicast GET over coaps to get the actual data, if permitted, from the privacy-sensitive server. This mechanism keeps the latency and number of messages to a minimum.

2. Alternatives Considered

This section discusses why existing alternatives are not sufficient.

2.1. Just use normal multicast discovery

Normal multicast discovery is susceptible to the threats discussed earlier. Another approach would be for multicast discovery to return only generic information that is the same for every device, and hence does not reveal any privacy related information or allow fingerprinting. This is undesirable since the resource handler would have to return different information based on whether the client is authenticated vs. unauthenticated, and thus is complex and error prone to implement and maintain.

2.2. Just use a resource directory

A resource directory could be used and only provide data to authenticated clients. However, the same problem still remains as to how to discover the resource directory itself. One could potentially use an alternate discovery protocol such as DNS-SD, but this introduces additional complexity when clients otherwise just use COAP for both discovery and communication. In addition, requiring a resource directory to be implemented, deployed, and maintained in a constrained environment presents an extra deployment burden that is desirable to avoid.

2.3. Use Alternative-Address

Section 4.5 of [I-D.ietf-core-coap-tcp-tls] provides an Alternative-Address option, which can be used to redirect the client to another transport address. However, it states:

The Alternative-Address elective option requests the peer to instead open a connection of the same kind as the present connection to the alternative transport address given. Its value is in the form "authority" as defined in Section 3.2 of [RFC3986].

Thus, Alternative-Address can indicate another authority component, but it explicitly requires the same URI scheme to be used, so it cannot be used to redirect from coap to coaps.

3. Redirects

3.1. Option Definitions

The following additional options are defined.

Number	Name	Format	Length	Base Value
TBD	Location-Scheme	string	0-255	(none)
TBD	Location-Authority	string	0-255	(none)

3.1.1. Location-Scheme and Location-Authority

Section 5.10.7 of [RFC7252] states:

The options that are used to compute the relative URI-reference are collectively called Location-* options. Beyond Location-Path and Location-Query, more Location-* options may be defined in the future and have been reserved option numbers 128, 132, 136, and 140.

The Location-Scheme and Location-Authority options are subject to all rules for Location-* options discussed in [RFC7252].

Together with Location-Path and Location-Query, the Location-Scheme and Location-Authority Options indicate a relative URI that contains either of an absolute path, a query string, or both. A combination of these options is included in a 3.01 (Moved Permanently) response to indicate the new location of the requested resource relative to the request URI.

If a response with Location-Scheme and/or Location-Authority Options passes through a cache that interprets these options and the implied URI identifies one or more currently stored responses, those entries MUST be marked as not fresh.

The Location-Scheme and Location-Authority Option can contain any character sequence conforming to the scheme and authority components defined in [RFC3986].

3.2. Response Codes

This specification adds the following response code.

3.2.1. 3.01 Moved Permanently

This Response Code indicates that the target resource has been assigned a new permanent URI and any future references to this resource ought to use the indicated effective URI.

The server MUST include in the response any of the following options whose values differ between the requested URI and the new effective URI: Location-Scheme, Location-Authority, Location-Path, and Location-Query. The client SHOULD use the Location field value for automatic redirection.

A 3.01 response is cacheable. Caches can use the Max-Age Option to determine freshness. A 3.01 response cannot be validated.

4. IANA Considerations

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

Number	Name	Reference
TBD	Location-Scheme	I-D.thaler-core-redirect
TBD	Location-Authority	I-D.thaler-core-redirect

NOTE: Section 5.10.7 of [RFC7252] reserves option numbers 128, 132, 136, and 140 for new Location-* options. Thus, the option numbers should be assigned from that set.

This document adds the following response codes to the "CoAP Response Codes" registry defined by [RFC7252]:

Code	Description	Reference
3.01	Moved Permanently	I-D.thaler-core-redirect

5. Security Considerations

The use case for this document is specifically to mitigate privacy concerns by allowing a request to an unsecured URI to be redirected to a secured URI.

Preventing identifying information from being observed by untrusted clients doing multicast discovery is necessary but not sufficient to mitigate the privacy issues discussed in Section 1. That is, one must also use an authentication scheme for subsequent unicast messages that does not reveal a stable identifier to clients before authentication is complete. Mutual authentication schemes exist (e.g., [Balfanz]) that only reveal the identity of both endpoints if authentication succeeds, but they may not yet be available in current standards and popular code bases.

6. References

6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://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, <<http://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, <<http://www.rfc-editor.org/info/rfc7252>>.

6.2. Informative References

- [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, <<http://www.rfc-editor.org/info/rfc6973>>.

- [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, <<http://www.rfc-editor.org/info/rfc7231>>.
- [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, <<http://www.rfc-editor.org/info/rfc7721>>.
- [I-D.ietf-core-coap-tcp-tls]
Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", draft-ietf-core-coap-tcp-tls-05 (work in progress), October 2016.
- [I-D.winfaa-intarea-broadcast-consider]
Winter, R., Faath, M., and F. Weisshaar, "Privacy considerations for IP broadcast and multicast protocol designers", draft-winfaa-intarea-broadcast-consider-03 (work in progress), September 2016.
- [Balfanz] Balfanz, D., Durfee, G., Shankar, N., Smetters, D., Staddon, J., and H-C. Wong, "Secret Handshakes From Pairing-based Key Agreements", May 2003, <<http://ieeexplore.ieee.org/document/1199336>>.
- [ListDiscussion]
Bormann, C., "Question about Location and redirection", Symposium on Security and Privacy 2003, October 2013, <<https://www.ietf.org/mail-archive/web/core/current/msg04867.html>>.
- [OIC1.1Core]
Open Connectivity Foundation, "OIC Core Specification V1.1.0", 2016, <https://openconnectivity.org/wp-content/uploads/2016/10/OIC_1.1-Specification.zip>.

Author's Address

Dave Thaler
Microsoft
One Microsoft Way
Redmond, WA 98052
USA

Email: dthaler@microsoft.com

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

M. Tiloca
RISE SICS AB
G. Selander
F. Palombini
Ericsson AB
J. Park
Universitaet Duisburg-Essen
October 27, 2017

Secure group communication for CoAP
draft-tiloca-core-multicast-oscoap-04

Abstract

This document describes a method for protecting group communication over the Constrained Application Protocol (CoAP). The proposed approach relies on Object Security for Constrained RESTful Environments (OSCORE) and the CBOR Object Signing and Encryption (COSE) format. All security requirements fulfilled by OSCORE are maintained for multicast OSCORE request messages and related OSCORE response messages. Source authentication of all messages exchanged within the group is ensured, by means of digital signatures produced through private keys of sender devices 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 <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on April 30, 2018.

Copyright Notice

Copyright (c) 2017 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
1.1. Terminology	3
2. Assumptions and Security Objectives	5
2.1. Assumptions	5
2.2. Security Objectives	7
3. OSCORE Security Context	7
3.1. Management of Group Keying Material	9
4. The COSE Object	10
5. Message Processing	12
5.1. Protecting the Request	12
5.2. Verifying the Request	13
5.3. Protecting the Response	13
5.4. Verifying the Response	13
6. Synchronization of Sequence Numbers	14
7. Security Considerations	14
7.1. Group-level Security	15
8. IANA Considerations	15
9. Acknowledgments	15
10. References	15
10.1. Normative References	15
10.2. Informative References	16
Appendix A. List of Use Cases	18
Appendix B. Example of Group Identifier Format	20
Appendix C. Set-up of New Endpoints	21
C.1. Join Process	21
C.2. Provisioning and Retrieval of Public Keys	23
C.3. Group Joining Based on the ACE Framework	24
Appendix D. Examples of Synchronization Approaches	25
D.1. Best-Effort Synchronization	25
D.2. Baseline Synchronization	25
D.3. Challenge-Response Synchronization	26
Appendix E. No Verification of Signatures	27
Authors' Addresses	28

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 A). 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 across intermediary nodes. To this end, a CoAP message is protected by including 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 multicast OSCORE, providing end-to-end security of CoAP messages exchanged between members of a multicast group. In particular, the described approach defines how OSCORE should be used in a group communication context, while fulfilling the same security requirements. That is, end-to-end security is assured for multicast CoAP requests sent by multicaster nodes to the group and for related CoAP responses sent as reply by multiple listener nodes. Multicast 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 multicaster node and a proxy (and vice versa), and between a proxy and a listener node (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]; 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", "Master Secret" and "Master Salt", 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 member of a multicast group. This includes, for instance, keys and IVs [RFC4949].
- o Group Manager (GM): entity responsible for creating a multicast group, establishing and provisioning Security Contexts among authorized group members, as well as managing the joining of new group members and the leaving of current group members. A GM can be responsible for multiple multicast groups. Besides, a GM is not required to be an actual group member and to take part in the group communication. The GM is also responsible for renewing/ updating Security Contexts and related keying material in the multicast groups of its competence. Each endpoint in a multicast group securely communicates with the respective GM.
- o Multicaster: member of a multicast group that sends multicast CoAP messages intended for all members of the group. In a 1-to-N multicast group, only a single multicaster transmits data to the group; in an M-to-N multicast group (where M and N do not necessarily have the same value), M group members are multicasters. According to [RFC7390], any possible proxy entity is supposed to know about the multicasters in the group and to not perform aggregation of response messages. Also, every multicaster expects and is able to handle multiple response messages associated to a given multicast request message that it has previously sent to the group.
- o Listener: member of a multicast group that receives multicast CoAP messages when listening to the multicast IP address associated to the multicast group. A listener may reply back, by sending a

response message to the multicaster which has sent the multicast message.

- o Pure listener: member of a multicast group that is configured as listener and never replies back to multicastrs after receiving multicast messages.
- o Endpoint ID: identifier assigned by the Group Manager to an endpoint upon joining the group as a new member, unless configured exclusively as pure listener. The Group Manager generates and manages Endpoint IDs in order to ensure their uniqueness within a same multicast group. That is, within a single multicast group, the same Endpoint ID cannot be associated to more endpoints at the same time. Endpoint IDs are not necessarily related to any protocol-relevant identifiers, such as IP addresses.
- o Group request: multicast CoAP request message sent by a multicaster in the group to all listeners in the group through multicast IP, unless otherwise specified.
- 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 either by a different group member or by a non-group member.

2. Assumptions and Security Objectives

This section presents a set of assumptions and security objectives for the approach described in this document.

2.1. 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 multicaster and multiple listeners) and M-to-N (multiple multicastrs and multiple listeners) 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). For instance, in a typical lighting control use case, 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.

- o Multicast group size: security solutions for group communication should be able to adequately support different, possibly large, group sizes. Group size is the combination of the number of multicasters and listeners in a multicast group, with possible overlap (i.e. a multicaster may also be a listener at the same time). In the use cases mentioned in this document, the number of multicasters (normally the controlling devices) is expected to be much smaller than the number of listeners (i.e. the controlled devices). A security solution for group communication that supports 1 to 50 multicasters would be able to properly cover the group sizes required for most use cases that are relevant for this document. The total number of group members is expected to be in the range of 2 to 100 devices. Groups larger than that should be divided into smaller independent multicast groups, e.g. by grouping lights in a building on a per floor basis.
- o Establishment and management of Security Contexts: a Security Context must be established among the group members by the Group Manager which manages the multicast group. A secure mechanism must be used to generate, revoke and (re-)distribute keying material, multicast security policies and security parameters in the multicast 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 multicast group. The ciphersuite is specified as part of the Security Context.
- o Backward security: a new device joining the multicast 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 multicast 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.

2.2. 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 multicast group SHALL be encrypted if privacy sensitive data is exchanged within the group. In fact, some control commands and/or associated responses could pose unforeseen security and privacy risks to the system users, when sent as plaintext. 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 multicast group, but not by an external adversary or other external entities.
- o Source authentication: messages sent within the multicast 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 (group authentication), and in particular by a specific member of the group (source authentication).
- o Message integrity: messages sent within the multicast 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.

3. OSCORE Security Context

To support multicast communication secured with OSCORE, each endpoint registered as member of a multicast group maintains a Security Context as defined in Section 3 of [I-D.ietf-core-object-security]. In particular, each endpoint in a group stores:

1. one Common Context, received from the Group Manager upon joining the multicast group and shared by all the endpoints in the group. All the endpoints in the group agree on the same COSE AEAD algorithm. In addition to what is defined in Section 3 of [I-D.ietf-core-object-security], the Common Context includes the following information.
 - * Group Identifier (Gid). Variable length byte string identifying the Security Context and used as Master Salt parameter in the derivation of keying material. The Gid is used together with the multicast IP address of the group to retrieve the Security Context, upon receiving a secure multicast request message (see Section 5.2). The Gid associated to a multicast group is determined by the responsible Group Manager. The choice of the Gid for a given group's Security Context is application specific. However, a Gid MUST be random as well as long enough, in order to achieve a negligible probability of collisions between Group Identifiers from different Group Managers. It is the role of the application to specify how to handle possible collisions. An example of specific formatting of the Group Identifier that would follow this specification is given in Appendix B.
 - * Counter signature algorithm. Value identifying 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 Security Context is established. All the endpoints in the group agree on the same counter signature algorithm. The Group Manager MUST define a list of supported signature algorithms as part of the group communication policy. Such a list MUST include the EdDSA signature algorithm ed25519 [RFC8032].
2. one Sender Context, unless the endpoint is configured exclusively as pure listener. The Sender Context is used to secure outgoing messages and is initialized according to Section 3 of [I-D.ietf-core-object-security], once the endpoint has joined the multicast group. In practice, the sender endpoint shares the same symmetric keying material stored in the Sender Context with all the recipient endpoints receiving its outgoing OSCORE messages. The Sender ID in the Sender Context coincides with the Endpoint ID received upon joining the group. It is responsibility of the Group Manager to assign Endpoint IDs to new joining endpoints in such a way that uniqueness is ensured within the multicast group. 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 messages are received, used to process such incoming secure messages. The endpoint creates a new Recipient Context upon receiving an incoming message from another endpoint in the group for the first time. In practice, the recipient endpoint shares the symmetric keying material stored in the Recipient Context with the associated other endpoint from which secure 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 secure messages are received.

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.

It is RECOMMENDED that the Group Manager acts as trusted key repository, and hence is configured to store public keys of group members and provide them to other members of the same group upon request. Possible approaches to provision public keys upon joining the group and to retrieve public keys of group members are discussed in Appendix C.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].

3.1. Management of Group Keying Material

The approach described in this specification should take into account the risk of compromise of group members. Such a risk is reduced when multicast groups are deployed in physically secured locations, like lighting inside office buildings. Nevertheless, 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 Section 2, 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 multicast group. The Group Manager responsible for the group is entrusted with such a task.

In particular, the Group Manager MUST distribute also a new Group Identifier (Gid) for that group, together with a new value for the Master Secret parameter. An example of how this can be done is provided in Appendix B. Then, each group member re-derives the keying material stored in its own Sender Context and Recipient Contexts as described in Section 3, using the updated Group Identifier.

Especially in dynamic, large-scale, multicast 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.

4. 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 responses SHALL be set to the Sender ID of the endpoint transmitting the group message.
- o The "unprotected" field of the "Headers" field SHALL additionally include the following parameters:
 - * gid : its value is set to the Group Identifier (Gid) of the group's Security Context. This parameter MAY be omitted if the message is a CoAP response.
 - * countersign : its value is set to the counter signature of the COSE object (Appendix C.3.3 of [RFC8152]), computed by the endpoint by means of its own private key as described in Section 4.5 of [RFC8152].

In particular, "gid" is included as COSE header parameter as defined in Figure 1.

name	label	value type	value registry	description
gid	TBD	bstr		Identifies the OSCORE group Security Context

Figure 1: Additional common header parameter for the COSE object

- o The Additional Authenticated Data (AAD) considered to compute the COSE object is extended, in order to include also the Group Identifier (Gid) of the Security Context used to protect the request message. In particular, the "external_aad" in Section 5.3 of [I-D.ietf-core-object-security] SHALL include also gid as follows:

```
external_aad = [
  version : uint,
  alg : int,
  request_kid : bstr,
  request_piv : bstr,
  gid : bstr,
  options : bstr
]
```

- o The OSCORE compression defined in Section 8 of [I-D.ietf-core-object-security] is used, with the following additions for the encoding of the object-security option.
 - * The fourth least significant bit of the first byte of the object-security option value SHALL be set to 1, to indicate the presence of the "kid" parameter for both multicast requests and responses.
 - * The fifth least significant bit of the first byte MUST be set to 1 for multicast requests, to indicate the presence of the Context Hint in the OSCORE payload. The Context Hint flag MAY be set to 1 for responses.
 - * The sixth least significant bit of the first byte is set to 1 if the "countersign" 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 Context Hint 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 "countersign" parameter including the counter signature of the COSE object.
- * The remaining bytes in the Object-Security value encode the value of the "kid" parameter, which is always present both in multicast requests and in responses.

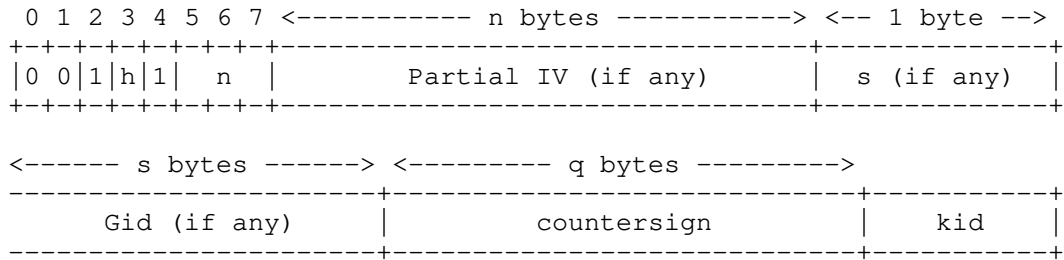


Figure 2: Object-Security Value

5. Message Processing

Each multicast 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.

Furthermore, endpoints in the multicast 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 4, without sending back any error message. This prevents listener endpoints from sending multiple error messages to a multicaster endpoint, so avoiding the risk of flooding the multicast group.

5.1. Protecting the Request

A multicaster endpoint transmits a secure multicast request message as described in Section 7.1 of [I-D.ietf-core-object-security], with the following modifications.

1. The multicaster endpoint stores the association Token - Group Identifier. That is, it SHALL be able to find the correct Security Context used to protect the multicast request and verify the response(s) by using the CoAP Token used in the message exchange.

2. The multicaster computes the COSE object as defined in Section 4 of this specification.

5.2. Verifying the Request

Upon receiving a secure multicast request message, a listener endpoint proceeds as described in Section 7.2 of [I-D.ietf-core-object-security], with the following modifications.

1. The listener endpoint retrieves the Group Identifier from the "gid" parameter of the received COSE object. Then, it uses the Group Identifier together with the destination IP address of the multicast request message to identify the correct group's Security Context.
2. The listener endpoint retrieves the Sender ID from the "kid" parameter of the received COSE object. Then, the Sender ID is used to retrieve the correct Recipient Context associated to the multicaster endpoint and used to process the request message. When receiving a secure multicast CoAP request message from that multicaster endpoint for the first time, the listener endpoint creates a new Recipient Context, initializes it according to Section 3 of [I-D.ietf-core-object-security], and includes the multicaster endpoint's public key.
3. The listener endpoint retrieves the corresponding public key of the multicaster endpoint from the associated Recipient Context. Then, it verifies the counter signature and decrypts the request message.

5.3. Protecting the Response

A listener endpoint that has received a multicast request message may reply with a secure response message, which is protected as described in Section 7.3 of [I-D.ietf-core-object-security], with the following modifications.

1. The listener endpoint computes the COSE object as defined in Section 4 of this specification.

5.4. Verifying the Response

Upon receiving a secure response message, a multicaster endpoint proceeds as described in Section 7.4 of [I-D.ietf-core-object-security], with the following modifications.

1. The multicaster endpoint retrieves the Security Context by using the Token of the received response message.

2. The multicaster endpoint retrieves the Sender ID from the "kid" parameter of the received COSE object. Then, the Sender ID is used to retrieve the correct Recipient Context associated to the listener endpoint and used to process the response message. When receiving a secure CoAP response message from that listener endpoint for the first time, the multicaster endpoint creates a new Recipient Context, initializes it according to Section 3 of [I-D.ietf-core-object-security], and includes the listener endpoint's public key.
3. The multicaster endpoint retrieves the corresponding public key of the listener endpoint from the associated Recipient Context. Then, it verifies the counter signature and decrypts the response message.

The mapping between response messages from listener endpoints and the associated multicast request message from a multicaster endpoint relies on the 3-tuple (Group ID, Sender ID, Partial IV) associated to the secure multicast request message. This is used by listener endpoints as part of the Additional Authenticated Data when protecting their own response message, as described in Section 4.

6. Synchronization of Sequence Numbers

Upon joining the multicast group, new listeners are not aware of the sequence number values currently used by different multicasters to transmit multicast request messages. This means that, when such listeners receive a secure multicast request from a given multicaster for the first time, they are not able to verify if that request is fresh and has not been replayed. The same applies when a listener endpoint loses synchronization with sequence numbers of multicasters, for instance after a device reboot.

The exact way to address this issue depends on the specific use case and its synchronization requirements. The Group Manager should define also how to handle synchronization of sequence numbers, as part of the policies enforced in the multicast group. In particular, the Group Manager can suggest to single specific listener endpoints how they can exceptionally behave in order to synchronize with sequence numbers of multicasters. Appendix D describes three possible approaches that can be considered.

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 multicast 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 multicast 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 multicast group belong to a common authority and are configured by a commissioner. For instance, in a professional lighting scenario, the roles of multicaster and listener are configured by the lighting commissioner, and devices strictly follow those roles.

8. IANA Considerations

TBD. Header parameter 'gid'.

9. Acknowledgments

The authors sincerely thank Stefan Beck, Rolf Blom, Carsten Bormann, Klaus Hartke, Richard Kelsey, John Mattsson, Jim Schaad and Ludwig Seitz for their feedback and comments.

10. References

10.1. 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-06 (work in
progress), October 2017.
- [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.amsuess-core-repeat-request-tag]
Amsuess, C., Mattsson, J., and G. Selander, "Repeat And Request-Tag", draft-amsuess-core-repeat-request-tag-00 (work in progress), July 2017.
- [I-D.aragon-ace-ipsec-profile]
Aragon, S., Tiloca, M., and S. Raza, "IPsec profile of ACE", draft-aragon-ace-ipsec-profile-00 (work in progress), July 2017.
- [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-01 (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)", draft-ietf-ace-oauth-authz-08 (work in progress), October 2017.
- [I-D.seitz-ace-oscoap-profile]
Seitz, L., Palombini, F., and M. Gunnarsson, "OSCORE profile of the Authentication and Authorization for Constrained Environments Framework", draft-seitz-ace-oscoap-profile-06 (work in progress), October 2017.

- [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 of OSCOAP multicast
groups in ACE", draft-tiloca-ace-oscoap-joining-00 (work
in progress), July 2017.
- [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](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](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](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](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. 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 Section 2.

- 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 multicast 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 multicast 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. 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.
- 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 multicast 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 multicast group. Furthermore, 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 multicast 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.

Appendix B. 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 uniquely defined in the set of all the multicast groups associated to the same Group Manager. The choice of the Group Prefix for a given group's Security Context is application specific. Group Prefixes are random as well as long enough, in order to achieve a negligible probability of collisions between Group Identifiers from different Group Managers.

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 3.1). 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 (see Section 3).

Appendix C. Set-up of New Endpoints

An endpoint joins a multicast group by explicitly interacting with the responsible Group Manager. All communications between a joining endpoint and the Group Manager rely on the CoAP protocol and MUST be secured. Specific details on how to secure communications between joining endpoints and a Group Manager are out of the scope of this specification.

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 maintain a number of information elements stored in its own Security Context (see Section 3). The following Appendix C.1 describes which of this information is provided to an endpoint upon joining a multicast group through the responsible Group Manager.

C.1. Join Process

An endpoint requests to join a multicast group by sending a confirmable CoAP POST request to the Group Manager responsible for that group. The join request is addressed to a CoAP resource associated to that group and carries the following information.

- o Role: the exact role of the joining endpoint in the multicast group. Possible values are: "multicaster", "listener", "pure listener", "multicaster and listener", or "multicaster and pure listener".
- 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 multicast group under its control. This

information is also omitted if the joining endpoint is configured exclusively as pure listener for the joined group. Appendix C.2 discusses additional details on provisioning of public keys and other information to enforce source authentication of joining node's messages.

- o Retrieval flag: indication of interest to receive the public keys of the endpoints currently in the multicast group, as included in the following join response. This flag MUST be set to false if the Group Manager is not configured to store the public keys of group members, or if the joining endpoint is configured exclusively as pure listener for the joined group.

The Group Manager MUST be able to verify that the joining endpoint is authorized to become a member of the multicast 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 C.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 node, before proceeding with the rest of the join process. Instead, in case the authorization check fails, the Group Manager MUST abort the join process. Further details about the authorization of joining endpoint are out of the scope of this specification.

As discussed in Section 3.1, it is then RECOMMENDED that the Security Context is renewed before the joining endpoint becomes a new active member of the multicast 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 multicast group, the Group Manager replies to the joining endpoint with a CoAP response carrying the following information.

- o Security Common Context: the OSCORE Security Common Context associated to the joined multicast group (see Section 3).
- o Endpoint ID: the Endpoint ID associated to the joining node. This information is not included in case "Role" in the join request is equal to "pure listener".
- 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 3.1). 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 node 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.

- o Member public keys: the public keys of the endpoints currently present in the multicast group. This includes: the public keys of the non-pure listeners currently in the group, if the joining endpoint is configured (also) as multicaster; and the public keys of the multicasters currently in the group, if the joining endpoint is configured (also) as listener or pure listener. 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 set to false in the join request. Appendix C.2 discusses additional details on provisioning public keys upon joining the group and on retrieving public keys of group members.

C.2. Provisioning and Retrieval of Public Keys

As mentioned in Section 3, it is RECOMMENDED that the Group Manager acts as trusted key repository, stores public keys of group members and provide 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 multicast group (see Appendix C.1).

After that, the Group Manager MUST verify that the joining endpoint actually owns the associated private key, for instance by performing a proof-of-possession challenge-response. In case of success, the Group Manager stores the received public key as associated to the joining endpoint and its Endpoint ID, before sending the join response and continuing with the rest of the join process. From then on, that public key will be available for secure and trusted delivery to other endpoints in the multicast group.

The joining node does not have to provide its own public key if that already occurred upon previously joining the same or a different multicast group under the same Group Manager. However, separately for each multicast group under its control, the Group Manager maintains an updated list of active Endpoint IDs associated to a same endpoint's public key.

Instead, in case the Group Manager does not act as trusted key repository, the following information is exchanged with the Group Manager during the join process.

1. The joining endpoint signs its own certificate by using its own private key. There is no restriction on the Certificate Subject included in the joining node's certificate.
2. The joining endpoint includes the following information as "Identity credentials" in the join request (Appendix C.1): the signed certificate; and the identifier of the Certification Authority that issued the certificate. The joining endpoint can optionally specify also a list of public key repositories storing its own certificate.
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 node.
4. The Group Manager stores the association between the Certificate Subject of the joining node's certificate and the pair {Group ID, Endpoint ID of the joining node}. 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 multicast 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.

C.3. Group Joining Based on the ACE Framework

The join process to register an endpoint as a new member of a multicast 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 [I-D.seitz-ace-oscoap-profile][I-D.ietf-ace-dtls-authorize][I-D.aragon-ace-ipsec-profile] 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 Client;
- o The Group Manager acts as Resource Server, with different CoAP resources for different multicast 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 multicast groups to join.

Both the joining endpoint and the Group Manager MUST 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 D. Examples of Synchronization Approaches

This section describes three possible approaches that can be considered by listener endpoints to synchronize with sequence numbers of multicasters.

D.1. Best-Effort Synchronization

Upon receiving a multicast request from a multicaster, a listener endpoint does not take any action to synchronize with the sequence number of that multicaster. This provides no assurance at all as to message freshness, which can be acceptable in non-critical use cases.

D.2. Baseline Synchronization

Upon receiving a multicast request from a given multicaster for the first time, a listener endpoint initializes its last-seen sequence number in its Recipient Context associated to that multicaster. However, the listener drops the multicast request without delivering it to the application layer. This provides a reference point to

identify if future multicast requests from the same multicaster are fresher than the last one received.

A replay time interval exists, between when a possibly replayed message is originally transmitted by a given multicaster and the first authentic fresh message from that same multicaster is received. This can be acceptable for use cases where listener endpoints admit such a trade-off between performance and assurance of message freshness.

D.3. Challenge-Response Synchronization

A listener endpoint performs a challenge-response exchange with a multicaster, by using the Repeat Option for CoAP described in Section 2 of [I-D.amsuess-core-repeat-request-tag].

That is, upon receiving a multicast request from a particular multicaster for the first time, the listener processes the message as described in Section 5.2 of this specification, but, even if valid, does not deliver it to the application. Instead, the listener replies to the multicaster with a 4.03 Forbidden response message including a Repeat Option, and stores the option value included therein.

Upon receiving a 4.03 Forbidden response that includes a Repeat Option and originates from a verified group member, a multicaster MUST send a group request as a unicast message addressed to the same listener, echoing the Repeat Option value. In particular, the multicaster 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 multicaster to not retain previously sent group requests for full retransmission, unless the application explicitly requires otherwise. In either case, the multicaster uses the sequence number value currently stored in its own Sender Context. If the multicaster stores group requests for possible retransmission with the Repeat Option, it should not store a given request for longer than a pre-configured time interval. Note that the unicast request echoing the Repeat Option is correctly treated and processed as a group message, since the "gid" field including the Group Identifier of the OSCORE group is still present in the Object-Security Option as part of the COSE object (see Section 4).

Upon receiving the unicast group request including the Repeat Option, the listener verifies that the option value equals the stored and previously sent value; otherwise, the request is silently discarded. Then, the listener verifies that the unicast group request has been received within a pre-configured time interval, as described in [I-D.amsuess-core-repeat-request-tag]. In such a case, the request

is further processed and verified; otherwise, it is silently discarded. Finally, the listener updates the Recipient Context associated to that multicaster, by setting the Replay Window according to the Sequence Number from the unicast group request conveying the Repeat Option. The listener either delivers the request to the application if it is an actual retransmission of the original one, or discard 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 group request including the Repeat Option within the configured time interval, the listener node SHOULD perform the same challenge-response upon receiving the next multicast request from that same multicaster.

A listener SHOULD NOT deliver group request messages from a given multicaster to the application until one valid group request from that same multicaster has been verified as fresh, as conveying an echoed Repeat Option [I-D.amsuess-core-repeat-request-tag]. Also, a listener MAY perform the challenge-response described above at any time, if synchronization with sequence numbers of multicasters 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 multicaster and the sequence number of a group request just received from the same multicaster. A multicaster MUST always be ready to perform the challenge-response based on the Repeat Option in case a listener starts it.

Note that endpoints configured as pure listeners 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 multicaster. Therefore, pure listeners should adopt alternative approaches to achieve and maintain synchronization with sequence numbers of multicasters.

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 multicast groups where many nodes at the same time might join as new members or lose synchronization.

Appendix E. No Verification of Signatures

There are some application scenarios using group communications 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 multicast 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.

Authors' Addresses

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

Email: marco.tiloca@ri.se

Goeran Selander
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: goran.selander@ericsson.com

Francesca Palombini
Ericsson AB
Torshamnsgatan 23
Kista SE-16440 Stockholm
Sweden

Email: francesca.palombini@ericsson.com

Jiye Park
Universitaet Duisburg-Essen
Schuetzenbahn 70
Essen 45127
Germany

Email: ji-ye.park@uni-due.de

June 2019

Expires: December 2019

Identity Modules for CoAP
draft-urien-core-identity-module-coap-06.txt

Abstract

This document defines identity modules based on Secure Elements processing DTLS/TLS stacks for CoAP devices. The expected benefits of these secure microcontrollers are the following :

- Secure storage of pre-share keys or private keys
- Trusted simple or mutual authentication between CoAP devices and CoAP clients.
- The device identity is enforced by a non cloneable chip.
- Trusted cryptographic support.
- Low power consumption for DTLS/TLS processing.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Status of this Memo

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

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

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

This Internet-Draft will expire on December 2019.

.

Copyright Notice

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

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

Table of Contents

Abstract.....	1
Requirements Language.....	1
Status of this Memo.....	1
Copyright Notice.....	2
1 Overview.....	4
2 What is a Secure Element.....	4
3 Identity Module for CoAP.....	6
4 DTLS/TLS profile for CoAP security modules.....	6
5 IANA Considerations.....	6
6 References.....	7
6.1 Normative References.....	7
6.2 Informative References.....	7
7 Authors' Addresses.....	7

1 Overview

The CoAP [RFC7252] protocol MAY be secured by the DTLS protocol [DTLS] over an UDP/IP stack; the TLS support [TLS] is defined by [RFC8323] over a TCP/IP stack.

According to [RFC7252] four security modes are available, NoSec, PreSharedKey, RawPublicKey, and Certificate. When DTLS is used with the PreShareKey or Certificate modes there is a need to store secrets such as symmetric or asymmetric keys, which authenticate the CoAP device.

In that case a Secure Element (SE) MAY be used in order to fully run the DTLS or TLS protocol. According to the data throughput or other security considerations the DTLS/TLS session MAY be exported from the secure element after the exchange of the finished messages.

This class of Secure Element is referred by this draft as an identity module (IdMod).

The expected benefits of identity modules are the following :

- Secure storage of pre-share keys or private keys
- Trusted simple or mutual authentication between the CoAP device and the CoAP client.
- The device identity is enforced by a non cloneable identity module.
- Trusted cryptographic support.
- Low power consumption for DTLS/TLS processing.

2 What is a Secure Element

A Secure Element (SE) is a tamper resistant microcontroller (see figure 1) equipped with host interfaces such as [ISO7816], SPI (Serial Peripheral Interface) or I2C (Inter Integrated Circuit).

The typical area size of these electronic chips is about 5x5 mm². They comprise CPU (8, 16, 32 bits), ROM (a few hundred KB), non volatile memory (EEPROM, FLASH, a few hundred KB) and RAM (a few ten KB). Security is enforced by multiple hardware and logical countermeasures.

According to the [EUROSMART] association height billion of such secure devices were shipped in 2013. Secure elements are widely deployed for electronic payment (EMV cards), telecommunication (SIM modules), identity (electronic passports), ticketing, and access control (PKCS15 cards).

Most of secure elements include a Java Virtual Machine (JVM) and therefore are able to execute embedded program written in the JAVACARD language. Because these devices are dedicated to security

purposes they support numerous cryptographic resources such as digest functions (MD5, SHA1, SHA2...), symmetric cipher (3xDES, AES) or asymmetric procedures (RSA, ECC).

A set of Global Platform [GP] standards control the lifecycle of embedded software, i.e. application downloading, activation and deletion.

As an illustration a typical low cost Secure Element has the following characteristics:

- JAVACARD operating system;
- Compliant with the GP (Global Platform) standards;
- 160 KB of ROM;
- 72 KB of EEPROM;
- 4KB of RAM;
- Embedded crypto-processor;
- 3xDES, AES, RSA, ECC;
- Certification according to Common Criteria (CC) EAL5+ level;
- Security Certificates from payment operators.

According to the state of art, TLS/DTLS stacks may run in secure elements, for example written as a javacard applications.

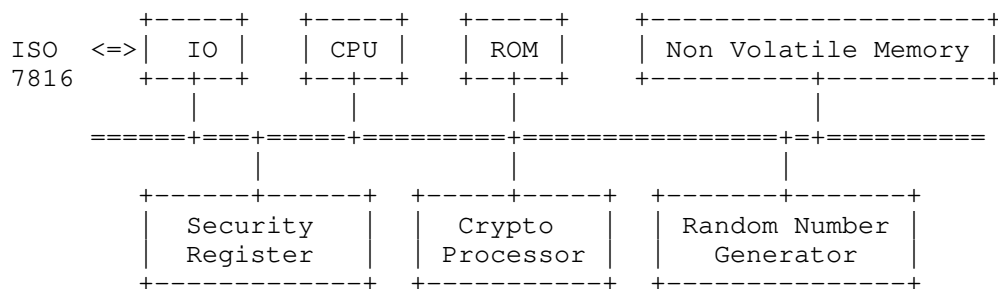


Figure 1. A typical hardware architecture of a Secure Element

3 Identity Module for CoAP

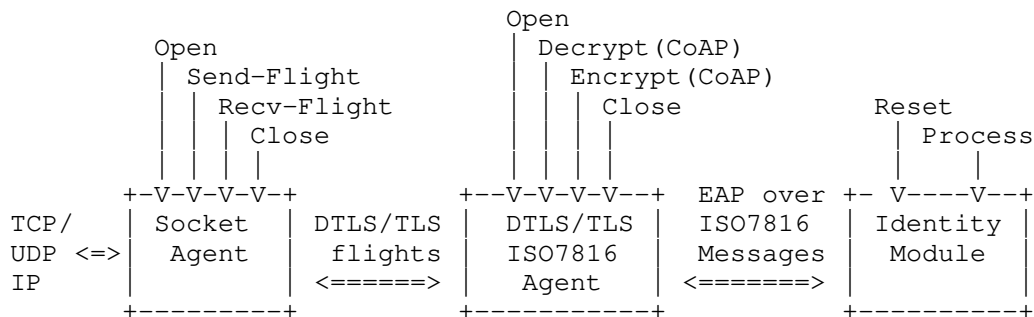


Figure 2. CoAP Identity module framework

ISO7816 interface for Secure Elements providing TLS/DTLS stacks are detailed in [DTLS/TLS-SM]. The Identity module MUST support two commands Reset and Process.

TLS/DTLS packets are transported by the EAP protocol over ISO7816 messages. This mechanism previously detailed by [EAPSC] provides a double segmentation procedure thanks to EAP and ISO7816 facilities.

A DTLS/TLS-ISO7816 software agent sends and receives DTLS/TLS flights to/from sockets over EAP/ISO7816 messages to/from the identity module. Conceptually this component interface SHOULD have four procedures Open, Close, Encrypt and Decrypt.

A socket software agent extracts and send DTLS/TLS flights from/to UDP/TCP packets. Conceptually this component interface SHOULD have four procedures Open, Close, Recv-Flight, Send-Flight.

4 DTLS/TLS profile for CoAP security modules

To be done.

5 IANA Considerations

This draft does not require any action from IANA.

6 References

6.1 Normative References

[TLS] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 5746, August 2008

[DTLS] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, January 2012.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, June 2014.

[RFC8323] CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets, February 2018.

[ISO7816] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO).

6.2 Informative References

[GP] Global Platform Standards, <http://www.globalplatform.org>

[EUROSMART] The EUROSMART association, <http://www.eurosmart.com>

[DTLS/TLS-SM] Urien, P., "TLS and DTLS Security Modules", draft-urien-uta-tls-dtls-security-module-08.txt, June 2019

[EAPSC] Urien, P., "EAP Support in Smartcard", draft-urien-eap-smartcard-37.txt, June 2019

7 Authors' Addresses

Pascal Urien
Telecom ParisTech
23 avenue d'Italie
75013 Paris
France

Phone: NA
Email: Pascal.Urien@telecom-paristech.fr

CORE Working Group
Internet Draft
Intended status: Experimental

P. Urien
Telecom Paris

April 3 2022

Expires: October 2022

Remote APDU Call Secure (RACS)
draft-urien-core-racs-16.txt

Abstract

This document describes the Remote APDU Call Protocol Secure (RACS) protocol, dedicated to Grid of Secure Elements (GoSE). These servers host Secure Elements (SE), i.e. tamper resistant chips offering secure storage and cryptographic resources.

Secure Elements are microcontrollers whose chip area is about 25mm²; they deliver trusted computing services in constrained environments.

RACS supports commands for GoSE inventory and data exchange with secure elements. It is designed according to the representational State Transfer (REST) architecture. RACS resources are identified by dedicated URIs. An HTTP interface is also supported.

An open implementation [OPENRACS] is available (<https://github.com/purien>) for various OS.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

Status of this Memo

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

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

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

This Internet-Draft will expire on October 2022.

.

Copyright Notice

Copyright (c) 2022 IETF Trust and the persons identified as the document authors. All rights reserved.

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

Table of Contents

Abstract.....	1
Requirements Language.....	1
Status of this Memo.....	1
Copyright Notice.....	2
1 Overview.....	5
1.1 What is a Secure Element.....	5
1.2 Grid Of Secure Elements (GoSE).....	6
1.3 Secure Element Identifier (SEID).....	7
1.3.1 SlotID example	7
1.3.2 SEID for Secure Elements	8
1.4 APDUs.....	9
1.4.1 ISO7816 APDU request	9
1.4.2 ISO7816 APDU response	9
2 The RACS protocol.....	10
2.1 Structure of RACS request.....	10
2.2 Structure of a RACS response.....	11
2.2.1 BEGIN Header	11
2.2.2 END Header	11
2.2.3 Status line	11
2.2.4 Examples of RACS responses:	12
2.3 RACS request commands.....	12
2.3.1 BEGIN	12
2.3.2 END	12
2.3.3 The APPEND parameter	13
2.3.4 GET-VERSION	14
2.3.5 SET-VERSION	14
2.3.6 LIST	15
2.3.7 RESET	15
2.3.8 APDU	16
2.3.9 SHUTDOWN	19
2.3.10 POWERON	20
2.3.11 ECHO	21
2.3.12 SEN	21
2.3.13 GET-SEN	23
2.4 Status header encoding.....	24
2.4.1 Event class	24
2.4.2 Command class	24
3 URI for the GoSE.....	25
4 HTTP interface.....	25
4.1 HTTPS Request.....	25
4.2 HTTPS response.....	26
5 Security Considerations.....	26
5.1 Authorization.....	26
5.2 Secure Element access.....	26
5.3 Applications security policy.....	27
5.3.1 Users-Table	27
5.3.2 SEID-Table	27
5.3.3 APDU-Table	27
5.4 Overview of the security policy.....	28
6 IANA Considerations.....	28

7	References.....	28
	7.1 Normative References.....	28
	7.2 Informative References.....	28
8	Authors' Addresses.....	29

1 Overview

This document describes the Remote APDU Call Protocol Secure (RACS) protocol, dedicated to Grids of Secure Elements (GoSE). These servers host Secure Elements (SE), i.e. tamper resistant chips offering secure storage and cryptographic resources.

Secure Elements are microcontrollers whose chip area is about 25mm²; they deliver trusted computing services in constrained environments.

RACS supports commands for GoSE inventory and data exchange with secure elements.

RACS is designed according to the representational State Transfer (REST) architecture [REST], which encompasses the following features:

- Client-Server architecture.
- Stateless interaction.
- Cache operation on the client side.
- Uniform interface.
- Layered system.
- Code On Demand.

1.1 What is a Secure Element

A Secure Element (SE) is a tamper resistant microcontroller equipped with host interfaces such as [ISO7816], SPI (Serial Peripheral Interface) or I2C (Inter Integrated Circuit).

The typical area size of these electronic chips is about 25mm². They comprise CPU (8, 16, 32 bits), ROM (a few hundred KB), nonvolatile memory (EEPROM, FLASH, a few hundred KB) and RAM (a few ten KB). Security is enforced by multiple hardware and logical countermeasures.

According to the [EUROSMART] association height billion of such secure devices were shipped in 2013. Secure elements are widely deployed for electronic payment (EMV cards), telecommunication (SIM modules), identity (electronic passports), ticketing, and access control.

Most of secure elements include a Java Virtual Machine and therefore are able to execute embedded program written in the JAVACARD language. Because these devices are dedicated to security purposes they support numerous cryptographic resources such as digest functions (MD5, SHA1, SHA2...), symmetric cipher (3xDES, AES) or asymmetric procedures (RSA, ECC).

A set of Global Platform [GP] standards control the lifecycle of embedded software, i.e. application downloading, activation and deletion.

As an illustration a typical Secure Element has the following characteristics:

- JAVACARD operating system;
- Compliant with the GP (Global Platform) standards;
- 160 KB of ROM;
- 72 KB of EEPROM;
- 4KB of RAM;
- Embedded crypto-processor;
- 3xDES, AES, RSA, ECC;
- Certification according to Common Criteria (CC) EAL5+ level;
- Security Certificates from payment operators.

1.2 Grid Of Secure Elements (GoSE)

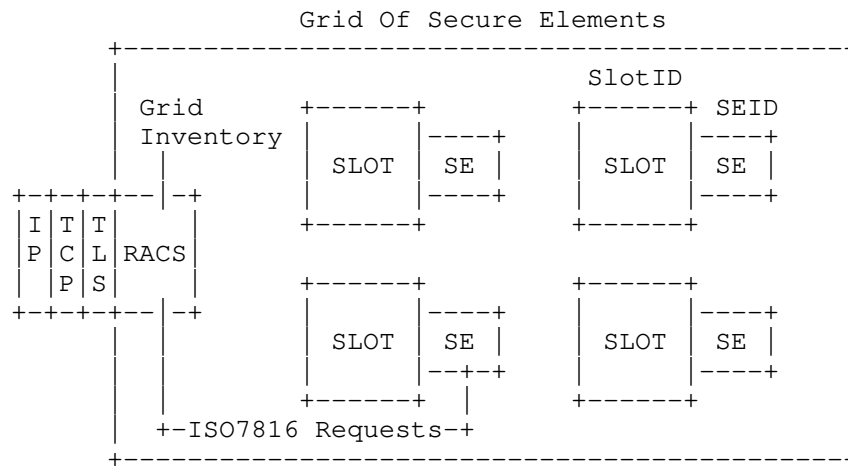


Figure 1. Architecture of a Grid of Secure Elements

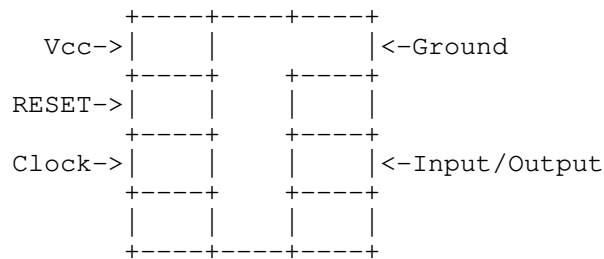


Figure 2. Illustration of an ISO7816 Secure Element

A grid of Secure Elements (GoSE) is a server hosting a set of secure elements.

The goal of these platforms is to deliver trusted services over the Internet. These services are available in two functional planes,

- The user plane, which provides trusted computing and secure storage.
- The management plane, which manages the lifecycle (downloading, activation, deletion) of applications hosted by the Secure Element.

A grid of Secure Elements offers services similar to HSM (Hardware Secure Module), but may be managed by a plurality of administrators, dealing with specific secure microcontrollers.

According to this draft all accesses to a GoSE require the TCP transport and are secured by the TLS [TLS 1.0] [TLS 1.1] [TLS 2.0] protocol.

The RACS protocol provides all the features needed for the remote use of secure elements, i.e.

- Inventory of secure elements
- Information exchange with the secure elements

1.3 Secure Element Identifier (SEID)

Every secure element needs a physical slot that provides electrical feeding and communication resources. This electrical interface is for example realized by a socket soldered on an electronic board, or a CAD (Card Acceptance Device, i.e. a reader) supporting host buses such as USB.

Within the GoSE each slot is identified by a SlotID (slot identifier) attribute, which may be a socket number or a CAD name.

The SEID (Secure Element Identifier) is a unique identifier indicating that a given SE is hosted by a GoSE. It also implicitly refers the physical slot (SlotID) to which the SE is plugged.

The GoSE manages an internal table that establishes the relationship between SlotIDs and SEIDs.

Therefore three parameters are needed for remote communication with secure element, the IP address of the GoSE, the associated TCP port, and the SEID.

1.3.1 SlotID example

According to the PC/SC (Personal Computer/Smart Card) standard [PS/SC], a smart card reader MAY include a serial number. This attribute (VENDOR-IFD-SERIAL) is associated to the tag 0x0103 in the class VENDOR-INFO.

1.3.2 SEID for Secure Elements

According to the Global Platform standard [GP] the Issuer Security Domain (ISD) manages applications lifecycle (downloading, activation, deletion). The command 'initialize update' is used to start a mutual authentication between the administration entity and the secure element; it collects a set of data whose first ten bytes are called the 'key diversification data'. This information is used to compute symmetric keys, and according for example to [EMV] MAY comprise a serial number.

1.4 APDUs

According to the [ISO7816] standards secure element process ISO7816 request messages and return ISO7816 response messages, named APDUs (application protocol data unit).

1.4.1 ISO7816 APDU request

An APDU request comprises two parts: a header and an optional body.

The header is a set of four or five bytes noted CLA INS P1 P2 P3

- CLA indicates the class of the request, and is usually bound to standardization committee (00 for example means ISO request).
- INS indicates the type of request, for example B0 for reading or D0 for writing.
- P1 P2 gives additional information for the request (such index in a file or identifier of cryptographic procedures)
- P3 indicates the length of the request body (from P3=01 to P3=FF), or the size of the expected response body (a null value meaning 256 bytes). Short ISO7816 requests may comprise only 4 bytes
- The body may be empty. Its maximum size is 255 bytes

1.4.2 ISO7816 APDU response

An APDU response comprises two parts an optional body and a mandatory status word.

- The optional body is made of 256 bytes at the most.
- The response ends by a two byte status noted SW. SW1 refers the most significant byte and SW2 the less significant byte.

An error free operation is usually associated to the 9000 status word. Following are some interpretations of the tuple SW1, SW2 according to various standards:

- '61' 'xx', indicates that xx bytes (modulus 256) are ready for reading. Operation result MUST be fetched by the ISO Get Response APDU (CLA=00, INS=C0, P1=P2=00, P3=XX)
- '9F' 'xx', indicates that xx bytes (modulus 256) are ready for reading. Operation result MUST be fetched by the ISO Get Response APDU (CLA=00, INS=C0, P1=P2=00, P3=XX)
- '6C' 'XX', the P3 value is wrong, request must be performed again with the LE parameter value sets to 'XX'
- '6E' 'XX', wrong instruction class (CLA) given in the request
- '6D' 'XX', unknown instruction code (INS) given in the request
- '6B' 'XX', incorrect parameter P1 or P2
- '67' 'XX', incorrect parameter P3
- '6F' 'XX', technical problem, not implemented...

2 The RACS protocol

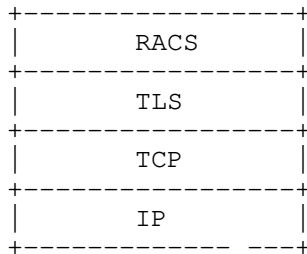


Figure 2. The RACS stack

The RACS protocol works over the TCP transport layer and is secured by the TLS protocol. The TLS client (i.e. the RACS client) MUST be authenticated by a certificate.

One of the main targets of the RACS protocol is to efficiently push a set of ISO7816 requests towards a secure element in order to perform cryptographic operations in the user's plane. In that case a RACS request typically comprises a prefix made with multiple ISO7816 requests and a suffix that collects the result of a cryptographic procedure.

The mandatory use of TLS with mutual authentication based on certificate provides a simple and elegant way to establish the credentials of a RACS client over the GoSE. It also enables an easy splitting between users' and administrators' privileges.

2.1 Structure of RACS request

A RACS request is a set of command lines, encoded according to the ASCII format. Each line ends by the Cr (carriage return) and line feed (Lf) characters. The RACS protocol is case sensitive.

Each command is a set of tokens (i.e. words) separated by space (0x20) character(s).

The first token of each line is the command to be executed.

A command line MAY comprise other tokens, which are called the command parameters.

A RACS request MUST start by a BEGIN command and MUST end by an END command.

Each command line is associated to an implicit line number. The BEGIN line is associated to the zero line number.

The processing of a RACS request is stopped after the first error. In that case the returned response contained the error status induced by the last executed command.

2.2 Structure of a RACS response

A RACS response is a set of lines, encoded according to the ASCII format. Each line ends by the Cr (carriage return) and line feed (Lf) characters. The RACS protocol is case sensitive.

Each line is a set of tokens (i.e. words) separated by space (0x20) character(s).

The first token of each line is the header.

The second token of response each line is associated command line number

A response line MAY comprise other tokens, which are called the response parameters.

Three classes of headers are defined BEGIN, END and Status.

A RACS response MUST start by a BEGIN header and MUST end by an END header. It comprises one or several status lines.

2.2.1 BEGIN Header

This header starts a response message.

It comprises an optional parameter, an identifier associated to a previous request message.

2.2.2 END Header

This header ends a response message.

2.2.3 Status line

A status header indicates a status line.

It begins by the character '+' in case of success or '-' if an error occurred during the RACS request execution. It is followed by an ASCII encoded integer, which is the value of the status.

The second mandatory token of a status line is the command line number (starting from zero)

A status line MAY comprise other tokens, which are called the response parameters.

2.2.4 Examples of RACS responses:

```
BEGIN CrLf
+001 000 Success CrLf
END CrLf
```

```
BEGIN moon1969 CrLf
-301 007 Illegal command, BEGIN condition not satisfied at line 7
END CrLf
```

```
BEGIN Asterix237 CrLf
+006 001 [ISO7816-Response] CrLf
END CrLf
```

```
BEGIN CrLf
-100 002 Unknown command at line 2 CrLf
END CrLf
```

```
BEGIN CrLf
-606 001 Unauthorized command APDU command at line 1
END CrLf
```

```
BEGIN CrLf
-706 001 SEID Already in use, APDU command at line 1
END CrLf
```

2.3 RACS request commands

2.3.1 BEGIN

This command starts a request message. A response message is returned if an error is detected.

An optional parameter is the request identifier, which MUST be echoed in the parameter of the first response line (i.e. starting by the BEGIN header).

2.3.2 END

This command ends a request message. It returns the response message triggered by the last command.

Example1

=====

Request:

BEGIN CrLf

END CrLf

Response:

BEGIN CrLf

+001 000 Success CrLf

END CrLf

Example2

=====

Request:

BEGIN Marignan1515 CrLf

APDU ASTERIX-CRYPTO-MODULE [ISO7816-Request] CrLf

END CrLf

Response:

BEGIN Marignan1515 CrLf

+006 001 [ISO7816-Response] CrLf

END CrLf

2.3.3 The APPEND parameter

The APPEND parameter MAY be used in all command lines, excepted BEGIN and END. The APPEND parameter MUST be the last parameter of a command line.

By default a response message returns only the last status line. When APPEND is inserted, the command line, if executed, MUST produce a status line.

Example

Request:

BEGIN SanchoPanza CrLf

APDU 100 [ISO7816-Request-1] CrLf

APDU 100 [ISO7816-Request-2] CrLf

END CrLf

Response:

BEGIN SanchoPanza CrLf

+006 002 [ISO7816-Response-2] CrLf

END CrLf

Request:

BEGIN DonQuichotte CrLf

APDU 100 [ISO7816-Request-1] APPEND CrLf

APDU 100 [ISO7816-Request-2] APPEND CrLf

END CrLf

Response:

```
BEGIN DonQuichotte CrLf
+006 001 [ISO7816-Response-1] CrLf
+006 002 [ISO7816-Response-2] CrLf
END CrLf
```

2.3.4 GET-VERSION

This command requests the current version of the RACS protocol. The returned response is the current version encoded by two integer separated by the '.' character. The first integer indicates the major version and the second integer gives the minor version.

This draft version is 0.2

Example

=====

Request:

```
BEGIN CrLf
GET-VERSION CrLf
END CrLf
```

Response:

```
BEGIN CrLf
+002 001 1.0 CrLf
END CrLf
```

2.3.5 SET-VERSION

This command sets the version to be used for the RACS request. An error status is returned by the response if an error occurred.

Example 1

=====

Request:

```
BEGIN CrLf
SET-VERSION 2.0 CrLf
END CrLf
```

Response:

```
BEGIN CrLf
-403 001 Error line 1 RACS 2.0 is not supported CrLf
END CrLf
```

Example 2

=====

Request:

```
BEGIN CrLf
SET-VERSION 1.0 CrLf
END CrLf
```

```
Response:
BEGIN CrLf
+003 001 RACS 1.0 has been activated CrLf
END CrLf
```

2.3.6 LIST

This command requests the list of SEID plugged in the GoSE.

It returns a list of SEIDs separated by space (0x20) character(s).

Some SEID attributes MAY be built from a prefix and an integer suffix (such as SE#100 in which SE# is the suffix and 100 is the integer suffix. A list of non-consecutive SEID MAY be encoded as prefix[i1;i2;...;ip] where i1,i2,ip indicates the integer suffix. A list of consecutive SEID could be encoded as prefix[i1-ip] where i1,i2,ip indicates the integer suffix.

Example 1

=====

```
Request:
BEGIN CrLf
LIST CrLf
END CrLf
```

```
Response:
BEGIN CrLf
+004 001 SEID1 SEID2 CR LF
END CrLf
```

Example 2

=====

```
Request:
BEGIN CrLf
LIST CrLf
END CrLf
```

```
Response:
BEGIN CrLf
+004 001 Device[1000-2000] SerialNumber[567;789;243] CrLf
END CrLf
```

2.3.7 RESET

This command resets a secure element. The first parameter gives the secure element identifier (SEID). An optional second parameter specifies a warm reset. The default behavior is a cold reset. The response status indicates the success or the failure of this operation.

Syntax: RESET SEID [WARM] CrLf

Example 1

=====

Request:

BEGIN CrLf

RESET device#45 CrLf

END CrLf

Response:

BEGIN CrLf

+005 001 device#45 Reset Done

END CrLf

Example 2

=====

Request:

BEGIN CrLf

RESET device#45 CrLf

END CrLf

Response:

BEGIN CrLf

-705 001 error device#45 is already in use

END CrLf

Example 3

=====

Request:

BEGIN CrLf

RESET device#45 WARM CrLf

END CrLf

Response:

BEGIN CrLf

+005 001 device#45 Warm Reset Done CrLf

END CrLf

2.3.8 APDU

This command sends an ISO7816 request to a secure element or a set of ISO7816 commands.

The first parameter specifies the SEID.

The second parameter is an ISO7816 request.

Three optional parameters are available; they MUST be located after the second parameter.

- CONTINUE=value, indicates that the next RACS command will be executed only if the ISO7816 status word (SW) is equal to a given value. Otherwise an error status is returned.
- MORE=value, indicates that a FETCH request will be performed (i.e. a new ISO7816 request will be sent) if the first byte of the ISO7816 status word (SW1) is equal to a given value.
- FETCH=value fixes the four bytes of the ISO7816 FETCH request (i.e. CLA INS P1 P2). The default value (when FETCH is omitted) is 00C00000 (CLA=00, INS=C0, P1=00, P2=00)

When the options CONTINUE and MORE are simultaneously set the SW1 byte is first checked. If there is no match then the SW word is afterwards checked.

The ISO7816 6Cxx status MUST be autonomously processed by the GoSE.

SYNTAX

APDU SEID ISO7816-REQUEST [CONTINUE=SW] [MORE=SW1] [FETCH=CMD] CrLf

The returned response is the ISO7816 response. If multiple ISO7816 requests are executed (due to the MORE option), the bodies are concatenated in the response, which ends by the last ISO7816 status word.

The pseudo code of the APDU command is the following :

```

1. BODY = empty;
2. SW   = empty;
3. DoIt = true;
3. Do
4. { iso7816-response = send(iso7816-request);
5.   body || sw1 || sw2 = iso7816-response;
6.   If ( (first request) && (iso7816-request.size==5) &&
        (body==empty) && (sw1==6C) )
7.   { iso7816-request.P3 = sw2 ; }
6.   Else
7.   { SW = sw1 || sw2
8.     BODY = BODY || body;
9.     If (sw1 == MORE)
10.    { iso7816-request = FETCH || sw2 ; }
11.    Else
12.    { DoIt=false; }
13.  }
14. }
15. While (DoIt == true)

16. iso7816-response = BODY || SW ;
17. If (SW != CONTINUE) Error ;
18. Else
    No Error;
```

Example 1

=====

Request:

BEGIN CrLf

APDU SEID ISO7816-REQUEST CrLf

END CrLf

Response:

BEGIN CrLf

+006 001 ISO7816-RESPONSE CrLf

END CrLf

Example 2

=====

Request:

BEGIN CrLf

APDU SEID ISO7816-REQUEST CrLf

END CrLf

Response:

BEGIN CrLf

-706 001 error SEID is already used CrLf

END CrLf

Example 3

=====

Request:

BEGIN CrLf

APDU SEID ISO7816-REQUEST CrLf

END CrLf

Response:

BEGIN CrLf

-606 001 error access unauthorized access CrLf

END CrLf

Example 4

=====

BEGIN CrLf

APDU SEID ISO7816-REQUEST-1 CONTINUE=9000 CrLf

APDU SEID ISO7816-REQUEST-2 CrLf

END CrLf

Response:

BEGIN CrLf

+006 002 ISO7816-RESPONSE-2 CrLf

END CrLf

Example 5

=====

```
BEGIN CrLf
APDU SEID ISO7816-REQUEST-1 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQUEST-2 CrLf
END CrLf
```

Response:

```
BEGIN CrLf
-006 001 Request Error line 1 wrong SW CrLf
END CrLf
```

Example 6

=====

```
BEGIN CrLf
APDU SEID ISO7816-REQ-1 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQ-2 CONTINUE=9000 CrLf
APDU SEID ISO7816-REQ-3 CONTINUE=9000 MORE=61 FETCH=00C00000 CrLf
END CrLf
```

Response:

```
BEGIN CrLf
+006 003 ISO7816-RESP-3 CrLf
END CrLf
```

Multiple ISO7816 requests have been performed by the third APDU command according to the following scenario :

- the ISO7816-REQ-3 request has been forwarded to the secure element (SEID)
- the ISO 7816 response comprises a body (body-0) and a status word (SW-0) whose first byte is 0x61, and the second byte is SW2-0
- the FETCH command CLA=00, INS=00, P1=00, P2=00, P3=SW2-0 is sent to the secure element
- the ISO 7816 response comprises a body (body-1) and a status word (SW-1) set to 9000

The RACS response is set to
+006 003 body-0 || body-1 || SW-1 CrLf
where || indicates a concatenation operation.

2.3.9 SHUTDOWN

This command powers down a secure element. The first parameter gives the secure element identifier (SEID).

Syntax: SHUTDOWN SEID CrLf

Example

=====

Request:

```
BEGIN Goodbye CrLf
SHUTDOWN device#45 CrLf
END CrLf
```

Response:

```
BEGIN Goodbye CrLf
+007 001 device#45 has been powered down CrLf
END CrLf
```

2.3.10 POWERON

This command powers up a secure element. The first parameter gives the secure element identifier (SEID).

Syntax: POWERON SEID CrLf

Example 1

=====

Request:

```
BEGIN CrLf
POWERON device#45 CrLf
END CrLf
```

Response:

```
BEGIN CrLf
+008 001 device#45 Has been powered up CrLf
END CrLf
```

Example 2

=====

Request:

```
BEGIN CrLf
POWERON device#45 CrLf
END CrLf
```

Response:

```
BEGIN CrLf
-708 001 error device#45 is already in use CrLf
END CrLf
```

Example 3

=====

Request:

```
BEGIN CrLf
POWERON device#45 CrLf
END CrLf
```

Response:
BEGIN CrLf
-608 001 error unauthorized access CrLf
END CrLf

2.3.11 ECHO

This command echoes a token. The first parameter is the token (word) to be echoed by the response.

Syntax: ECHO SEID CrLf

Example 1
=====

Request:
BEGIN TestEcho CrLf
ECHO Hello CrLf
END CrLf

Response:
BEGIN TestEcho CrLf
+009 001 Hello CrLf
END CrLf

Example 2
=====

Request:
BEGIN ResetSEID CrLf
POWERON device#45 CrLf
ECHO Done CrLf
END CrLf

Response:
BEGIN ResetSEID CrLf
+009 001 Done CrLf
END CrLf

2.3.12 SEN

This command associates Secure Element Name (SEN) to SEID. Secure Element Name are defined in [IOSE]

The first parameter (mandatory) is the SEID. By default the SEN is found in the ISO7816 ATR, and the TLS-SE application is the secure element default application.

The second parameter (optional) is the SEN. This option sets the SEN, and discards the ATR content.

The third parameter (optional) is the TLS-SE Application Identifier (AID).

Syntax: SEN SEID [SEN] [AID] CrLf

Example 1

=====

Request:

BEGIN CrLf

SEN mySEID CrLf

END CrLf

Response:

BEGIN CrLf

+010 001 SEN= key1.com AID= default

END CrLf

Example 2

=====

Request:

BEGIN CrLf

SEN mySEID key1.com CrLf

END CrLf

Response:

BEGIN CrLf

+010 001 SEN= key1.com AID= default CrLf

END CrLf

Example 3

=====

Request:

BEGIN CrLf

SEN mySEID key1.com 010203040500 CrLf

END CrLf

Response:

BEGIN CrLf

+010 001 SEN= key1.com AID= 010203040500 CrLf

END CrLf

Example 4

=====

Request:
BEGIN CrLf
SEN wrongSEID key1.com CrLf
END CrLf

Response:
BEGIN CrLf
-410 001 SEN invalid SEID (wrongSEID) CrLf
END CrLf

2.3.13 GET-SEN

This command gets Secure Element Name (SEN) associated to SEID.
Secure Element Name are defined in [IOSE]

Syntax: GET-SEN SEID CrLf

Example 1 =====

Request:
BEGIN CrLf
GET-SEN mySEID CrLf
END CrLf

Response:
BEGIN CrLf
+011 001 key1.com [AID= default]
END CrLf

Example 2 =====

Request:
BEGIN CrLf
GET-SEN mySEID CrLf
END CrLf

Response:
BEGIN CrLf
+011 001 key1.com [AID= 010203040500]
END CrLf

Example 3 =====

Request:
BEGIN CrLf

```
GET-SEN wrongSEID CrLf
END CrLf
```

```
Response:
BEGIN CrLf
-511 001 GET-SEN invalid SEID (wrongSEID)
END CrLf
```

2.4 Status header encoding

The first token of a response line is the status header. It begins by a '+' or a '-' character, and comprises three decimal digits (xyz).

The first digit (x) MUST indicate an event class.
The second and third digits (yz) MAY indicate a command class.

2.4.1 Event class

This draft only defines the meaning of the first digit located at the left most side.

```
+0yz: No error
-0yz: Command execution error
-1yz: Unknown command, the command is not defined by this draft
-2yz: Not implemented command
-3yz: Illegal command, the command can't be executed
-4yz: Not supported parameter or parameter illegal value
-5yz: Parameter syntax error or parameter missing
-6yz: Unauthorized command
-7yz: Already in use, a session with this SE is already opened
-8yz: Hardware error
-9yz: System error
```

2.4.2 Command class

The second and third digits (yz) MAY indicate the command that triggered the current line status

```
01 BEGIN
02 GET-VERSION
03 SET-VERSION
04 LIST
05 RESET
06 APDU
07 SHUTDOWN
08 POWERON
09 ECHO
10 SEN
11 GET-SEN
```


3 URI for the GoSE

The URI addressing the resources hosted by the GoSE is represented by the string:

`RACS://GoSE-Name:port/?request`

where request is the RACS request to be forwarded to a the GoSE.

RACS command lines are encoded in a way similar to the INPUT field of an HTML form. Each command is associated to an INPUT name, the remaining of the command line i.e. a set of ASCII characters, is written according to the URL encoding rules. End of line characters, i.e. carriage return (Cr) and line feed (Lf) are omitted.

As a consequence a request is written to the following syntax
`cmd1=cmd1-parameters&cmd2=cmd2-parameters`

Example:

`RACS://GoSE-Name:port/?BEGIN=&APDU=SEID%20[ISO7816-REQUEST]&END=`

4 HTTP interface

A GoSE SHOULD support an HTTP interface. RACS requests/responses are transported by HTTP messages. The use of TLS is mandatory.

4.1 HTTPS Request

`https://GoSE-Name:port/RACS?request`

where request is the RACS request to be forwarded to a secure element (SEID)

The RACS request is associated to an HTML form whose name is "RACS". The request command lines are encoded as the INPUT field of an HTML form. Each command is associated to an INPUT name, the remaining of the command line i.e. a set of ASCII characters is written according to the URL encoding rules. End of line characters, i.e. carriage return (Cr) and line feed (Lf) are omitted.

As a consequence a RACS request is written as
`https://GoSE-Name/RACS?cmd1=cmd1-parameters&cmd2=cmd2-parameters`

Example:

`https://GoSE-Name/RACS?BEGIN=&APDU=SEID%20[ISO7816-REQUEST]&END=`

4.2 HTTPS response

The RACS response is returned in an XML document.

The root element of the document is <RACS-Response>

The optional parameter of the BEGIN header, is the content of the <begin> element.

Each status line is the content of the <Cmd-Response> element, which includes the following information :

- The status header is the content of the <status> element.
- The line number is the content of the <line> element.
- The other parameters of the status line are the content of the <parameters> element.

The END header is associated to the element <end>

End of line, i.e. carriage return (Cr) and line feed (Lf) characters are omitted.

As a consequence a RACS response is written as :

```
<RACS-Response>
<begin>Optionnal-ID</begin>
<Cmd-Response
<status>+000</status>
<line>001</line>
<parameters>other parameters of the RACS response</parameters>
</Cmd-Response>
<end></end>
</RACS-Response>
```

5 Security Considerations

5.1 Authorization

A RACS client MUST be authenticated by an X509 certificate.

The GoSE software MUST provide a mean to establish a list of SEIDs that can be accessed from a client whose identity is the CommonName (CN) attribute of its certificate. It MAY allocate a UserID (UID), i.e. an integer index from the certificate common name.

5.2 Secure Element access

The GoSE MUST manage a unique session identifier (SID) for each TLS session. The SID is bound to the client's certificate CommonName (SID(CN))

A secure element has two states, unlocked and locked. In the locked state the secure element may be only used by the SID that previously locked it.

The first authorized command that successfully accesses to a SEID (either POWERON ,RESET, APDU) locks a secure element (SEID) with the current session (SID).

The SHUTDOWN command MUST unlock a secure element (SEID).

The end of a TLS session MUST unlock all the secure elements locked by the session.

5.3 Applications security policy

According to the [ISO7816] standards each Application embedded within a secure element (associated to a SEID) is identified by an AID parameter (16 bytes at the most)

The RACS server SHOULD support the following facilities

5.3.1 Users-Table

Each CN (the Users-Table primary key) is associated to a list of SEIDs whose access is authorized.

5.3.2 SEID-Table

Each AID (the SEID-Table primary key) is associated to a list of CNs whose access is authorized.

5.3.3 APDU-Table

For a given AID and an authorized CN, an APDU-Table MAY be available. This table acts as a firewall, which defined a set of forbidden ISO7816 commands.

For example this filter could be expressed as a set of the four first bytes of an APDU-Prefix (CLA INS P1 P2) and a four bytes Mask
An ISO7816-Request is firewall if:

ISO7816-Request AND Mask IsEQUAL to APDU-Prefix

5.4 Overview of the security policy

The summary of the security policy is illustrated by the figure 3.

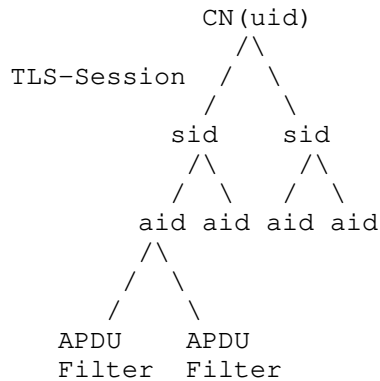


Figure 3. Summary of the security policy

6 IANA Considerations

This draft does not require any action from IANA.

7 References

7.1 Normative References

[TLS 1.0] Dierks, T., C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999

[TLS 1.1] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006

[TLS 1.2] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5746, August 2008

[TLS 1.3] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, August 2018

[ISO7816] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO)

7.2 Informative References

[REST] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", 2000, <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>

[GP] Global Platform Standards, <http://www.globalplatform.org>

[EUROSMART] The EUROSMART association, <http://www.eurosmart.com>

[PC/SC] The PC/SC workgroup, <http://www.pcscworkgroup.com>

[EMV] EMV Card Personalization Specification, Version 1.1, July 2007

[OPENRACS] <https://github.com/purien>, open RACS implementation for Win32, Ubuntu, Raspberrypi

[IOSE] Internet of Secure Elements, draft-urien-coinrg-iose-03.txt, September 2021

8 Authors' Addresses

Pascal Urien
Telecom Paris
19 place Marguerite Perey
91120 Palaiseau Phone: NA
France Email: Pascal.Urien@telecom-paris.fr

anima
Internet-Draft
Intended status: Standards Track
Expires: May 2, 2017

S. Kumar
Philips Lighting Research
P. van der Stok
Consultant
October 29, 2016

EST based on DTLS secured CoAP (EST-coaps)
draft-vanderstok-core-coap-est-00

Abstract

Low-resource devices in a Low-power and Lossy Network (LLN) can operate in a mesh network using the IPv6 over Low-power Personal Area Networks (6LoWPAN) and IEEE 802.15.4 link-layer standards. Provisioning these devices in a secure manner with keys (often called security bootstrapping) used to encrypt and authenticate messages is the subject of Bootstrapping of Remote Secure Key Infrastructures (BRSKI) [I-D.ietf-anima-bootstrapping-keyinfra]. Enrollment over Secure Transport (EST) [RFC7030], based on TLS and HTTP, is used for BRSKI. This document defines how low-resource devices are expected to use EST over DTLS and CoAP. 6LoWPAN fragmentation management and minor extensions to CoAP are needed to enable EST over DTLS-secured CoAP (EST-coaps).

Note

Many of the concepts in this document are taken over from [RFC7030]. Consequently, much text is directly traceable to [RFC7030]. The same document structure is followed to point out the differences and commonalities between EST and EST-coaps.

Status of This Memo

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

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

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

This Internet-Draft will expire on May 2, 2017.

Copyright Notice

Copyright (c) 2016 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
1.1. Terminology	4
2. Operational Scenarios Overview	4
3. Protocol Design and Layering	5
3.1. CoAP response codes	7
3.2. Message fragmentation using Block	7
3.3. CoAP message headers	8
4. Protocol Exchange Details	9
5. IANA Considerations	9
6. Security Considerations	12
7. Acknowledgements	12
8. Change Log	12
9. References	12
9.1. Normative References	12
9.2. Informative References	13
Appendix A. Operational Scenario Example Messages	14
Authors' Addresses	15

1. Introduction

IPv6 over Low-power Wireless Personal Area Networks (6LoWPANs) [RFC4944] on IEEE 802.15.4 [ieee802.15.4] wireless networks is becoming common in many professional application domains such as lighting controls. However commissioning of such networks suffers from a lack of standardized secure bootstrapping mechanisms for these networks.

Although IEEE 802.15.4 defines how security can be enabled between nodes within a single mesh network, it does not specify the provisioning and management of the keys. Therefore securing a 6LoWPAN network with devices from multiple manufacturers with

different provisioning techniques is often tedious and time consuming.

Bootstrapping of Remote Secure Infrastructures (BRSKI) [I-D.ietf-anima-bootstrapping-keyinfra] addresses the issue of bootstrapping networked devices in the context of Autonomic Networking Integrated Model and Approach (ANIMA). However, BRSKI has not been developed specifically for low-resource devices in constrained networks. These networks use DTLS [RFC6347], CoAP [RFC7252], and UDP instead of TLS [RFC5246], HTTP [RFC7230] and TCP. BRSKI relies on Enrollment over Secure Transport (EST) [RFC7030] for the provisioning of the operational domain certificates. Replacing the EST invocations of TLS and HTTP by DTLS and CoAP invocations enables applying BRSKI on CoAP-based low-resource devices.

The Figure 1 below shows the EST-coaps architecture.

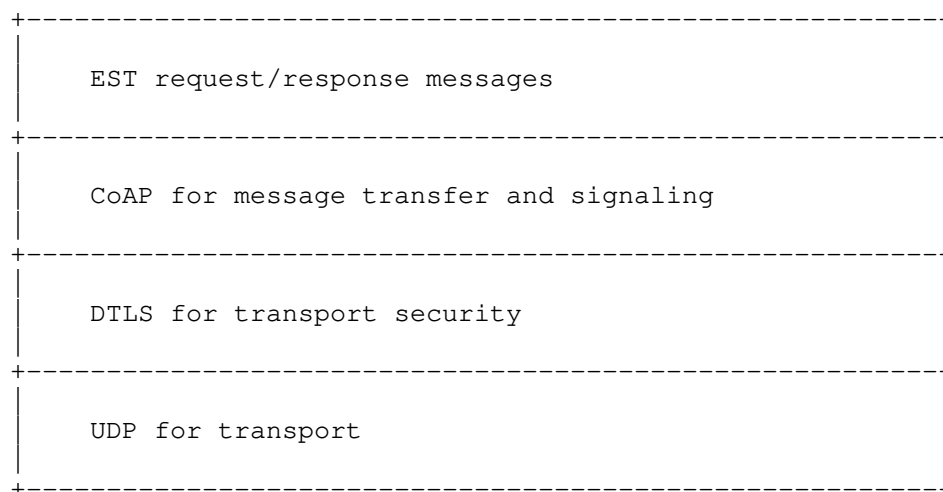


Figure 1: EST-coaps protocol layers

Although EST-coaps paves the way for the utilization of BRSKI for constrained devices on constrained networks, some devices will not have enough resources to handle the large payloads that come with EST-coaps. It is up to the network designer to decide which devices execute the BRSKI protocol and which not.

EST-coaps is designed for use in professional control networks such as lighting. The autonomic bootstrapping is interesting because it reduces the manual intervention during the commissioning of the

network. Typing in passwords is contrary to this wish. Therefore, the password authentication of EST is not supported in EST-coaps.

In the constrained devices context it is very unlikely that full PKI request messages will be used. For that reason, full PKI messages are not supported in EST-coaps.

Because the relatively large messages involved in EST cannot be readily transported over constrained (6LoWPAN, LLN) wireless networks, this document defines the use of CoAP Block-Wise Transfer ("Block") [RFC7959] combined with DTLS to fragment EST messages at the application layer.

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].

All the terminology from EST [RFC7030] is included in this document by reference.

2. Operational Scenarios Overview

Only the differences to EST with respect to operational scenarios are described in this section. EST-coaps server authentication differs from EST as follows:

- o Replacement of TLS by DTLS and HTTP by CoAP, resulting in:
 - * DTLS-secured CoAP sessions between EST-coaps client and EST-coaps server.
- o Only certificate-based client authentication is supported, with as result:
 - * The EST-coaps client does not support manual authentication (as described in Section 4.4.1 of [RFC7030])
 - * The EST-coaps client does not support authentication at the application layer.
- o EST-coaps does not support full PKI request messages [RFC5272].

The following EST-coaps protocol parts are supported as described for the equivalent EST parts:

1. Request of client certificates by submitting a enrollment request to EST-coaps server.
2. Renewal of existing client certificates by submitting a re-enrollment request to EST-coaps server.
3. Request of certificate with key pair generated by EST-coaps server.
4. The EST-coaps client can request the attributes needed for enrollment before the enrollment request is issued"

3. Protocol Design and Layering

The EST-coaps protocol design follows closely the EST design, excluding some aspects that are not relevant for automatic bootstrapping of constrained devices within a professional context. The parts supported by EST-coaps are:

Message types:

- * Simple PKI messages.
- * CA certificate retrieval.
- * CSR Attributes Request.
- * Server-generated key request.

CoAP with Block-Wise Transfer:

- * CoAP Block-Wise Transfer header Options for control of the transfer of larger EST messages.

DTLS for transport security:

- * Authentication of the EST-coaps server.
- * Authentication of the EST-coaps client.
- * Communication integrity and confidentiality.
- * Channel-binding information for linking proof-of-identity with message-based proof-of-possession (OPTIONAL).

Given that CoAP and DTLS can provide proof of identity for EST-coaps clients and server, simple PKI messages can be used conformant to section 3.1 of [RFC5272]. EST-coaps supports the certificate types

and Trust Anchors (TA) that are specified for EST in section 3 of [RFC7030].

The EST-coaps server URI is identical to the EST URI (except for replacing the scheme https by coaps):

```
coaps://www.example.com/.well-known/est
coaps://www.example.com/.well-known/est/arbitraryLabel1
```

See Figure 5 in section 3.2.2 of [RFC7030] for the path-suffixes (operations) that are supported by EST.

EST-coaps uses CoAP to transfer EST messages, aided by Block-Wise Transfer [RFC7959] to transport CoAP messages in blocks thus avoiding (excessive) 6LoWPAN fragmentation of UDP datagrams. The use of "Block" is specified in Section 3.2.

The content-format (media type equivalent) of the CoAP message determines which EST message is transported in the CoAP payload. The media types specified in the HTTP Content-Type header (see section 3.2.2 of [RFC7030]) are in EST-coaps specified by the Content-Format Option (12) of CoAP. The combination of URI path-suffix and content-format used MUST map to an allowed combination of path-suffix and media type as defined for EST.

EST-coaps is designed for use between low-resource devices using CoAP and hence does not need to send base64-encoded data. Simple binary coding is more efficient (30% less payload compared to base64) and well supported by CoAP. Therefore, the content formats specification in Section 5 requires the use of binary encoding for all EST-coaps CoAP payloads.

The functions of TLS specified for EST are in EST-coaps mapped to the equivalent DTLS functions. However, DTLS sessions SHOULD remain open for persistent EST-coaps connections to reduce storage load. For example, a cacerts request followed by an enrollments request SHOULD use the same DTLS session.

The mandatory cipher suite for DTLS is TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 defined in [RFC7251] which is the mandatory-to-implement cipher suite in CoAP. Additionally the curve secp256r1 MUST be supported [RFC4492]; this curve is equivalent to the NIST P-256 curve. The hash algorithm is SHA-256. DTLS implementations MUST use the Supported Elliptic Curves and Supported Point Formats Extensions [RFC4492]; the uncompressed point format MUST be supported; [RFC6090] can be used as an implementation method.

3.1. CoAP response codes

Section 5.9 of [RFC7252] specifies the mapping of HTTP response codes to CoAP response codes. Every time the HTTP response code 200 is specified in [RFC7030] in response to a GET request, in EST-coaps the equivalent CoAP response code 2.05 MUST be used. Response code HTTP 202 in EST is mapped as indicated below; while other HTTP 2xx response codes are not used by EST. For the following HTTP 4xx error codes that may occur: 400, 401, 403, 404, 405, 406, 412, 413, 415 ; the equivalent CoAP response code for EST-coaps is 4.xx. For the HTTP 5xx error codes: 500, 501, 502, 503, 504 the equivalent CoAP response code is 5.xx.

HTTP response code 202 needs a different treatment from the one described for [RFC7030]. A new CoAP response code 2.06 is needed. When the EST over CoAP request cannot be treated immediately, a CoAP response code 2.06 Delayed is returned with Content-Format: application/link-format described in [RFC6690]. The payload of the response contains a link to receive the delayed response. ALTERNATIVE (to discuss) : a 2.06 Delayed response without payload and the link to receive the delayed response indicated using the Location-Path and Location-Query Options.

The waiting client may send GET requests to the returned link. When the response is not available, the server returns response code 2.06 with again the link for the client to query. When the response is available, the server returns the response code 2.05 Content with a payload containing the requested response in the appropriate content format.

3.2. Message fragmentation using Block

DTLS defines fragmentation only for the handshake part and not for secure data exchange (DTLS records). [RFC6347] states "Each DTLS record MUST fit within a single datagram". In order to avoid using IP fragmentation, which is not supported by 6LoWPAN, invokers of the DTLS record layer MUST size DTLS records so that they fit within any Path MTU estimates obtained from the record layer. In addition, invokers residing on a 6LoWPAN over IEEE 802.15.4 network SHOULD attempt to size CoAP messages such that each DTLS record will fit within one or two IEEE 802.15.4 frames only by choosing the appropriate block sizes.

Certificates can vary greatly in size dependent on signature algorithms and key sizes. For a 256-bit curve, common ECDSA sizes fluctuate between 500 bytes and 1 KB. Some EST messages may be several kilobytes in size. Given non-existence of IP fragmentation in 6LoWPAN networks and its 1280 bytes MTU, EST-coaps needs to be

able to fragment EST messages into multiple DTLS datagrams with each DTLS datagram containing a block of CoAP payload data. Further considering the small payload size available to a CoAP message, which can be as low as 68 bytes in case the message needs to fit into a single IEEE 802.15.4 frame, fine-grained fragmentation of EST messages is essential.

For CoAP, [RFC7959] specifies the "Block1" option for fragmentation of the request payload and the "Block2" option for fragmentation of the return payload. The CoAP client MAY specify the Block1 size and MAY also specify the Block2 size. The CoAP server MAY specify the Block2 size, but not the Block1 size.

Examples of fragmented messages are shown in Appendix A.

3.3. CoAP message headers

EST-coaps uses CoAP payload blocks that each fit in a single DTLS record i.e. UDP datagram without causing IP fragmentation. The returned CoAP response codes are specified in Section 3.1. The CoAP Token value is not specified by EST-coaps and may be chosen by the CoAP client according to [RFC7252].

An example HTTP request message cacerts in EST will look like:

```
REQ:
      GET /.well-known/est/cacerts HTTP/1.1
      Host: 192.0.2.1:8085
      Accept: */*
```

```
RES:
      HTTP/1.1 200 OK
      Status: 200 OK
      Content-Type: application/pkcs7-mime
      Content-Transfer-Encoding : base64
      Content-Length: 4246
      payload
```

The corresponding EST-coaps request looks like:

```
REQ:
      GET coaps://[192.0.2.1:8085]/.well-known/est/cacerts

RES:
      2.05 Content (Content-Format: application/pkcs7-mime)
      {payload}
```

4. Protocol Exchange Details

The EST-coaps client MUST be configured with an implicit TA database or an explicit TA database. The authentication of the EST-coaps server by the EST-coaps client is based on Certificate authentication in the DTLS handshake.

The authentication of the EST-coaps client is based on client certificate in the DTLS handshake. This can either be

- o DTLS with a previously issued client certificate (e.g., an existing certificate issued by the EST CA);
- o DTLS with a previously installed certificate (e.g., manufacturer-installed certificate or a certificate issued by some other party);

The details on checking the validity of the certificates are identical to EST.

The other protocol aspects such as simple enrollment (re-enrollment), certificate attributes and CA certificate request are similar to EST with the exception that these are performed on coaps (CoAP+DTLS) as the transport. The required content-formats for these request and response messages are defined in Section 5. The CoAP response codes are defined in Section 3.1.

EST-coaps does not support full PKI Requests. Consequently, the fullcmc request of section 4.3 of [RFC7030] and response MUST NOT be supported by EST-coaps.

5. IANA Considerations

Additions to the sub-registry "CoAP Content-Formats", within the "CoRE Parameters" registry are needed for the below media types. These can be registered either in the Expert Review range (0-255) or IETF Review range (256-9999).

1.

- * application/pkcs7-mime
- * Type name: application
- * Subtype name: pkcs7-mime
- * smime-type: certs-only

- * ID: TBD1
- * Required parameters: None
- * Optional parameters: None
- * Encoding considerations: Binary
- * Security considerations: As defined in this specification
- * Published specification: [RFC5751]
- * Applications that use this media type: ANIMA Bootstrap (BRSKI) and EST

2.

- * application/pkcs8
- * Type name: application
- * Subtype name: pkcs8
- * ID: TBD2
- * Required parameters: None
- * Optional parameters: None
- * Encoding considerations: Binary
- * Security considerations: As defined in this specification
- * Published specification: [RFC5958]
- * Applications that use this media type: ANIMA Bootstrap (BRSKI) and EST

3.

- * application/csrattrs
- * Type name: application
- * Subtype name: csrattrs
- * ID: TBD3

- * Required parameters: None
- * Optional parameters: None
- * Encoding considerations: Binary
- * Security considerations: As defined in this specification
- * Published specification: [RFC7030]
- * Applications that use this media type: ANIMA Bootstrap (BRSKI) and EST

4.

- * application/pkcs10
- * Type name: application
- * Subtype name: pkcs10
- * ID: TBD4
- * Required parameters: None
- * Optional parameters: None
- * Encoding considerations: binary
- * Security considerations: As defined in this specification
- * Published specification: [RFC5967]
- * Applications that use this media type: ANIMA bootstrap (BRSKI) and EST

Additions to the sub-registry "CoAP Response Code", within the "CoRE Parameters" registry are needed for the following response codes:

- o Code: 2.06
- o Description: Delayed
- o Reference: this document

6. Security Considerations

The security considerations mentioned in EST applies also to EST-coaps.

7. Acknowledgements

The authors are very grateful to Klaus Hartke for his detailed explanations on the use of Block with DTLS. The authors would like to thank Esko Dijk and Michael Verschoor for the valuable discussions that helped in shaping the solution.

8. Change Log

9. References

9.1. Normative References

- [I-D.ietf-anima-bootstrapping-keyinfra]
Pritikin, M., Richardson, M., Behringer, M., and S. Bjarnason, "Bootstrapping Remote Secure Key Infrastructures (BRSKI)", draft-ietf-anima-bootstrapping-keyinfra-03 (work in progress), June 2016.
- [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>>.
- [RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <<http://www.rfc-editor.org/info/rfc4492>>.
- [RFC5272] Schaad, J. and M. Myers, "Certificate Management over CMS (CMC)", RFC 5272, DOI 10.17487/RFC5272, June 2008, <<http://www.rfc-editor.org/info/rfc5272>>.
- [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, <<http://www.rfc-editor.org/info/rfc5751>>.
- [RFC5958] Turner, S., "Asymmetric Key Packages", RFC 5958, DOI 10.17487/RFC5958, August 2010, <<http://www.rfc-editor.org/info/rfc5958>>.

- [RFC5967] Turner, S., "The application/pkcs10 Media Type", RFC 5967, DOI 10.17487/RFC5967, August 2010, <<http://www.rfc-editor.org/info/rfc5967>>.
- [RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <<http://www.rfc-editor.org/info/rfc6090>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <<http://www.rfc-editor.org/info/rfc6347>>.
- [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <<http://www.rfc-editor.org/info/rfc6690>>.
- [RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<http://www.rfc-editor.org/info/rfc7030>>.
- [RFC7251] McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS", RFC 7251, DOI 10.17487/RFC7251, June 2014, <<http://www.rfc-editor.org/info/rfc7251>>.
- [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>>.
- [RFC7959] Bormann, C. and Z. Shelby, Ed., "Block-Wise Transfers in the Constrained Application Protocol (CoAP)", RFC 7959, DOI 10.17487/RFC7959, August 2016, <<http://www.rfc-editor.org/info/rfc7959>>.

9.2. Informative References

- [ieee802.15.4]
Institute of Electrical and Electronics Engineers, , "IEEE Standard 802.15.4-2006", 2006.
- [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, <<http://www.rfc-editor.org/info/rfc4944>>.

- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <<http://www.rfc-editor.org/info/rfc6838>>.
- [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, <<http://www.rfc-editor.org/info/rfc7230>>.

Appendix A. Operational Scenario Example Messages

This appendix provides detailed examples of the messages using DTLS and BLOCK option Block2. The minimum PMTU is 1280 bytes, which is the example value assumed for the DTLS datagram size. The example block length is taken as 64 which gives an SZX value of 2.

The following is an example of a valid /cacerts exchange.

During the initial DTLS handshake, the client can ignore the optional server-generated "certificate request" and can instead proceed with the CoAP GET request. The content length of the cacerts response in appendix A.1 of [RFC7030] is 4246 bytes using base64. This leads to a length of 3185 bytes in binary. The CoAP message adds around 10 bytes, the DTLS record 29 bytes.

To avoid IP fragmentation, the CoAP block option is used and an MTU of 127 is assumed to stay within one IEEE 802.15.4 packet. To stay below the MTU of 127, the payload is split in 50 packets with a payload of 64 bytes each. Fifty times the client sends an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP Request. The server returns an IPv6 packet containing the UDP datagram with the DTLS record that encapsulates the CoAP response.

The CoAP request-response exchange with block option is shown below. Block option is shown in a decomposed way indicating the kind of Block option (2 in this case because used in the response) followed by a colon, and then the block number (NUM), the more bit (M = 0 means last block), and block size exponent (2*(SZX+4)) separated by slashes. The Length 64 is used with SZX= 2 to avoid IP fragmentation.

The CoAP Request is sent with confirmable (CON) option and the content format of the Response is /application/cacerts.

```
GET [192.0.2.1:8085]/.well-known/est/cacerts -->
    <-- (2:0/1/64) 2.05 Content
    GET URI (2:1/1/64) -->
        <-- (2:1/1/64) 2.05 Content
            |
            |
    GET URI (2:49/1/64) -->
        <-- (2:49/0/64) 2.05 Content
```

Authors' Addresses

Sandeep S. Kumar
Philips Lighting Research
High Tech Campus 7
Eindhoven 5656 AE
NL

Email: ietf@sandeep.de

Peter van der Stok
Consultant

Email: consultancy@vanderstok.org

CoRE
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2017

P. van der Stok, Ed.
consultant
J. Jimenez
Ericsson
October 27, 2016

Mapping from LWM2M model to CoMI YANG model
draft-vanderstok-core-yang-lwm2m-00

Abstract

This document defines a set of rules to convert a LWM2M xml-based device specification to a YANG MODULE. The invocation of the server executing the converted YANG code makes use of CoMI. The mapping from the original LWM2M URI to the corresponding CoMI URI is presented.

Note

Discussion and suggestions for improvement are requested, and should be sent to roll@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 <http://datatracker.ietf.org/drafts/current/>.

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

This Internet-Draft will expire on April 30, 2017.

Copyright Notice

Copyright (c) 2016 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
1.1. Terminology	3
1.1.1. Tree Diagrams	3
2. Conversion rules LWM2M to YANG	4
3. URI convention	7
4. observation and notification	8
5. Payload format	8
6. YANG extensions to LWM2M	8
7. Security considerations	8
8. Acknowledgements	8
9. Changelog	8
10. References	9
10.1. Normative References	9
10.2. Informative References	9
Appendix A. YANG identifiers as IDnumbers	10
Appendix B. YANG identifiers as resource names	13
Appendix C. YANG identifiers as additional leaf	16
Authors' Addresses	21

1. Introduction

Standardization organizations define interfaces hosted by processors to manipulate the connected equipment. Examples of such standardization organizations are BACnet, KNX, ZigBee, oBIX, OMA/IPSO, and many others. These organizations plan to move to resource based interfaces. The data models proposed by these organizations are hierarchical models that can be specified in XML and describe classes with attributes and operations that can be instantiated to objects. An example is the OMA LWM2M (see [OMNA]) Object model, that standardizes eight numbered object types for device management. IPSO (see [IPSO]) expands those objects to handle applications. This document describes rules to translate xml specifications of the LWM2M/IPSO organizations to YANG [RFC7950], and the invocation of the YANG based server according to the CoAP Management Interface (CoMI) specification [I-D.vanderstok-core-comi].

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 [RFC6241] and are not redefined here:

- o client
- o configuration data
- o server
- o state data

The following terms are defined in [RFC7950] and are not redefined here:

- o data model
- o data node

The terminology for describing YANG data models is found in [RFC7950].

1.1.1. Tree Diagrams

A simplified graphical representation of the data model is used in the YANG modules specified in this document. The meaning of the symbols in these diagrams is as follows:

Brackets "[" and "]" enclose list keys.

Abbreviations before data node names: "rw" means configuration data (read-write) and "ro" state data (read-only).

Symbols after data node names: "?" means an optional node, "!" means a presence container, and "*" denotes a list and leaf-list.

Parentheses enclose choice and case nodes, and case nodes are also marked with a colon (":").

Ellipsis ("...") stands for contents of subtrees that are not shown.

2. Conversion rules LWM2M to YANG

LWM2M objects are typed, where each type is identified with a number. The object provides one or more instances which are numbered. An instance is composed of resources, also identified with numbers. An instance on a host can be accessed with the example URI: `coap+lwm2m://example.com/object/instance/resource`, where resource, instance and object are numbers, specified by the LWM2M specification.

When using YANG, the object identifiers, followed by the resource identifier, are YANG strings instead of numbers. The format of the instance identifier depends on the YANG model that is chosen to specify LWM2M objects.

For an automatic translation from the XML LWM2M specification to a YANG specification, the following rules apply for access, optional, units, and range specifications:

- o The optional/mandatory aspect of the LWM2M resource is covered by the leaf's mandatory "false/true" statement of YANG as specified in section 7.6.5 of [RFC7950]. The YANG statement "mandatory = TRUE" means that (given the right conditions) the leaf must exist.
- o The R,W access aspects of a data item are translated using the YANG "config" statement as specified in section 7.21.1 of [RFC7950]. If "config" is "true", the Data nodes are part of configuration datastores, resulting in RW access. If "config" is "false", the data nodes are not part of configuration datastores, resulting in R access. YANG does not provide facilities to specify W access only.
- o When the YANG RPC is specified, E access is meant. In section 7.14 of [RFC7950] RPCs are modelled for NETCONF using YANG input and output parameters. When input parameters are added, EW access is meant; when output parameters are added, ER access is meant and with both input and output parameters ERW access is meant.
- o The YANG ACTION is specified in section 7.15 of [RFC7950]. Contrary to RPC, ACTION statement is associated with a data node. The data node has E access. Input leafs of the data node have W access, and output leafs have R access.
- o To specify the range of a data resource the YANG range statement, specified in section 9.2.4 of [RFC7950], is used.
- o YANG range can be used in a straightforward fashion for items of type integer. The range of decimal64, used for float, is less

straightforward. The possible ranges are restricted by the fraction-digits which specifies the size of the fraction part of the float (see section 9.3.4. of [RFC7950]).

- o The YANG units statement, specified in section 7.3.3 of [RFC7950], is used to express the units.
- o The attributes of the YANG leaf need to be presented in the order: "type", possibly qualified with "range", "units", "config", "mandatory", and finally "Description".
- o In the presented YANG specification the LWM2M resources are specified as leafs of a YANG list.

YANG lists may contain key leafs which uniquely identify an instance in a list. By specifying a key leaf (for example called "instance") that contains the list instance number, the YANG list instance can be uniquely referenced by the instance number. Accordingly, OMA objects are modelled as YANG lists. The value of the "instance" leaf in the list is equal to the instance number of the OMA object. The numbering of the instances does not need to be consecutive. The OMA resources are the other leafs of the YANG list.

Choices need to be made how to represent the numbered object ID, and resource ID as YANG identifiers. YANG identifiers are strings and cannot be represented by numbers.

The YANG identifier strings need to be mapped to numbered identifiers. The appendices show 3 ways to represent the LWM2M device ID and resource ID in the YANG specification.

- o In Appendix A, Yang Identifiers are modelled as strings that start with string "ID" followed by the identifier number (see module humidityID).
- o In Appendix B, Yang Identifiers of objects and resources are modelled as strings that are equivalent to the OMA object- and resource- name (see module humidityNM).
- o In Appendix C, the OMA device is modelled as a YANG container composed of an identifier and a list of instances. The list is composed of an instance number and a set of resource containers. The resource container is composed of the pair (attribute identifier number, IPSO resource specification) (see module humidityLF).

Below the tree diagrams (see Section 1.1.1 for an explanation of the syntax) of the three valid YANG modules are shown.

```

module: ietf-yang-humidityID
+--ro ID3304* [instance]
+--ro instance                               uint16
+--ro ID5700                                decimal64
+--ro ID5701?                               string
+--ro ID5601?                               decimal64
+--ro ID5602?                               decimal64
+--ro ID5603?                               decimal64
+--ro ID5604?                               decimal64
+---x ID5605

module: ietf-yang-humidityNM
+--ro IPSO-humidity* [instance]
+--ro instance                               uint16
+--ro Sensor_Value                          decimal64
+--ro Units?                               string
+--ro Min_Measured_Value?                   decimal64
+--ro Max_Measured_Value?                   decimal64
+--ro Min_Range_Value?                     decimal64
+--ro Max_Range_Value?                     decimal64
+---x Reset_Min_and_Max_measured_values

module: ietf-yang-humidityLF
+--rw IPSO-humidity
+--ro identifier    uint16
+--ro resources* [instance]
+--ro instance      uint16
+--ro Sensor_Value
|   +--ro identifier?    uint16
|   +--ro content        decimal64
+--ro Units
|   +--ro identifier?    uint16
|   +--ro content?       string
+--ro Min_Measured_Value
|   +--ro identifier?    uint16
|   +--ro content?       decimal64
+--ro Max_Measured_Value
|   +--ro identifier?    uint16
|   +--ro content?       decimal64
+--ro Min_Range_Value
|   +--ro identifier?    uint16
|   +--ro content?       decimal64
+--ro Max_Range_Value
|   +--ro identifier?    uint16
|   +--ro content?       decimal64
+--ro Reset_Min_and_Max_measured_values
+--ro identifier?       uint16

```

+---x reset

Module humidityLF of Appendix C is the most complex one and is not recommended. Module humidityID of Appendix A works but is a bit forced approach and lacks the resource name. Module humidityNM of Appendix B is the most natural approach where the YANG identifiers are equal to the device (type) and resource names.

CoMI [I-D.vanderstok-core-comi] uses a conversion from names to numbers to reduce the request URI size, and the payload of the server requests and answers. The LWM2M organization specifies both the names and the numbers of the devices and resources. The number of the resource is not unique and for the CoMI identifier the resource number needs to be prefixed by the device number to be unique.

3. URI convention

The invocation URI of a LWM2M resource looks like:

```
coap+lwm2m://example.com/object/instance/resource
```

In this section it is assumed that the YANG mapping of the module humidityNM of Appendix B is used.

When YANG is used, the LWM2M resource invocation can follow the RESTCONF convention using http, or the CoMI convention using CoAP.

When using RESTCONF (see section 3.5.3 of [I-D.ietf-netconf-restconf]) the invocation of object with instance = number will look like:

```
http://example.com/object/instance=number/resource
```

In the case of CoMI the object/resource numbers are used, and not the names, to reduce the payload size. The instance is specified in a query parameter. Consequently, the LWM2M resource on a server executing a YANG specification, is accessed according to the CoMI specification with:

```
coap://example.com/identifier?k=number
```

The identifier is a composition of the object number and the resource number. Assume that n is smallest number for which $10^{n+1}/\text{resource} \geq 1$. The value of the identifier = $(\text{object} * 10^n) + \text{resource}$.

Assume that the IPSO-humidity/Sensor_Value 3304/5700 numbers are composed to the numeric identifier 33045700. According to [RFC4648],

the identifier is represented in base64 which leads to B-DzE. The URI for the CoMI invocation of instance 0 of IPSO-humidity/Sensor_value will look like:

```
coap://example.com/B-DzE?k=0
```

For LWM2M objects with only one instance, the k=0 can be omitted.

4. observation and notification

An LWM2M server uses "observe" to receive notification from the server. This remains unchanged with YANG servers and CoMI.

5. Payload format

The payload of the request and the response follows the payload format specified for CoMI. The content format is CBOR [RFC7049]. The YANG objects are returned as maps containing (identifier, value) pairs. Where the identifier is the numeric identifier discussed in Section 3. and the value is of the type specified by the YANG specification of the server. The CBOR encoding of the YANG types is specified in [I-D.ietf-core-yang-cbor].

6. YANG extensions to LWM2M

By adding keys leafs to a list object, YANG allows additionally the selection of instances by the contents of the key leafs.

The FETCH method of CoAP makes it possible to request multiple resource instances in one request.

The notification statement of YANG encourages a more flexible specification of notifications.

7. Security considerations

To be filled in

8. Acknowledgements

We are grateful to

9. Changelog

NO changes from nothing to version 00

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, <<http://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, <<http://www.rfc-editor.org/info/rfc7950>>.
- [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, <<http://www.rfc-editor.org/info/rfc6241>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<http://www.rfc-editor.org/info/rfc4648>>.
- [RFC7049] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", RFC 7049, DOI 10.17487/RFC7049, October 2013, <<http://www.rfc-editor.org/info/rfc7049>>.
- [I-D.vanderstok-core-comi]
Stok, P. and A. Bierman, "CoAP Management Interface", draft-vanderstok-core-comi-09 (work in progress), March 2016.
- [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-02 (work in progress), July 2016.

10.2. Informative References

- [I-D.ietf-netconf-restconf]
Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-17 (work in progress), September 2016.
- [OMNA] "Open Mobile Naming Authority (OMNA)", Web <http://http://technical.openmobilealliance.org/Technical/technical-information/omna>.

[IPSO] "IP for Smart Objects (IPSO)",
Web <http://ipso-alliance.github.io/pub/>.

Appendix A. YANG identifiers as IDnumbers

Yang Identifiers are modelled as string that starts with ID followed by the identifier number. The device object is modelled as a list that contains multiple instances.

```
<CODE BEGINS> file "ietf-humidityID@2016-07-25.yang"
module ietf-humidityID{

    yang-version 1.1;  // needed for action

    namespace
    "urn:ietf:params:xml:ns:yang:ietf-humidityID";

    prefix humid;

    organization
    "IPSO";

    contact
    "WG Web:  http://tools.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:   Peter van der Stok
    mailto:consultancy@vanderstok.org

    Editor:   Jaime Jimenez
    mailto:jaime.jimenez@ericsson.com";

    description
    "This module contains information about the operation of the
    IPSO LWM2M humidity sensor with ID 3304.
```

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."

```
    revision "2016-07-25" {
      description "Initial revision.";
      reference
        "I-D:draft-vanderstok-core-yang-lwm2m: YANG language applied
        to the LWM2M IPSO humidity sensor specification";
    }

    list ID3304 {
      key instance;
      config false;      // should be same for key leaf
      description
        "IPSO humidity: The humidity sensor is composed of
        a set of instances";
      leaf instance {
        type uint16{
          range "0..1";    // only one instance zero (0)
        }
        config false;      // R access
        mandatory "true";
        description
          "the number of the humidity sensor instance";
      }
      leaf ID5700 {
        type decimal64{    // YANG has no float
          fraction-digits 2;
          range "10.0 .. 66.6";
        }
        config false;      // R access
        mandatory "true";
        description
          "Sensor Value: Last or Current Measured Value
          from the Sensor";
      }
      leaf ID5701 {
        type string;
        units "Defined by 'Units' resource";
        config false;      // R access
        description
          "Units: Measurement unit definition
          e.g. 'Cel' for temperature in Celsius";
      }
      leaf ID5601 {
```

```
type decimal64{    // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false;    // R access
description
"Min Measured Value: The minimum value measured
  by the sensor since power ON or reset";
}
leaf ID5602 {
type decimal64{    // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false;    // R access
description
"Max Measured Value: The maximum value measured
  by the sensor since power ON or reset";
}
leaf ID5603 {
type decimal64{    // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false;    // R access
description
"Min Range Value: The minimum value that
  can be measured by the sensor";
}
leaf ID5604 {
type decimal64{    // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false;    // R access
description
"Max Range Value: The maximum value that
  can be measured by the sensor";
}
action ID5605 {
//E access: this is an RPC
//    without input and output parameters
description
"Reset Min and Max measured values: Reset the
  Min and Max measured values to current value";
}
} // list ID3304
} // module ietf-yang-humidity
```


<CODE ENDS>

Appendix B. YANG identifiers as resource names

Yang Identifiers are modelled as strings that represent the resource name. The device object is modelled as a list with multiple instances.

```
<CODE BEGINS> file "ietf-humidityNM@2016-07-25.yang"

module ietf-humidityNM{

yang-version 1.1;  // needed for action

namespace
    "urn:ietf:params:xml:ns:yang:ietf-humidityNM";

prefix humid;

    organization
        "IPSO";

    contact
        "WG Web:  http://tools.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:  Peter van der Stok
mailto:consultancy@vanderstok.org

Editor:  Jaime Jimenez
mailto:jaime.jimenez@ericsson.com";

    description
        "This module contains information about the
operation of the IPSO LWM2M humidity sensor with ID 3304.
```

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.";

```
        revision "2016-07-25" {
            description "Initial revision.";
            reference
                "I-D:draft-vanderstok-core-yang-lwm2m:
                YANG language applied to the LWM2M IPSO humidity sensor
                specification";
        }

        list IPSO-humidity {
            key instance;
            config false;    // should be same as key leaf
            description
                "3304: The humidity sensor is composed of
                a set of instances";
            leaf instance {
                type uint16{
                    range "0..1";    // only one instance zero (0)
                }
                config false;    // R access
                mandatory "true";
                description
                    "the number of the humidity sensor instance";
            }
            leaf Sensor_Value {
                type decimal64{    // YANG has no float
                    fraction-digits 2;
                    range "10.0 .. 66.6";
                }
                units "Defined by 'Units' resource";
                config false;    // R access
                mandatory "true";
                description
                    "5700: Last or Current Measured Value
                    from the Sensor";
            }
            leaf Units {
                type string;
                units "Defined by 'Units' resource";
                config false;    // R access
                description
```

```
"5701: Measurement unit definition
    e.g. 'Cel' for temperature in Celsius";
}
leaf Min_Measured_Value {
type decimal64{    // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false;    // R access
description
"5601: The minimum value measured by
    the sensor since power ON or reset";
}
leaf Max_Measured_Value {
type decimal64{    // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false;    // R access
description
"5602: The maximum value measured
    by the sensor since power ON or reset";
}
leaf Min_Range_Value {
type decimal64{    // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false;    // R access
description
"5603: The minimum value that can be measured
    by the sensor";
}
leaf Max_Range_Value{
type decimal64{    // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false;    // R access
description
"5604: The maximum value that can be measured
    by the sensor";
}
action Reset_Min_and_Max_measured_values {
// E access: this is an RPC
// without input and output parameter
description
"5605: Reset the Min and Max measured values
```

```
        to current value";
    } // rpc
    } // list ID3304
    } // module ietf-yang-humidity

<CODE ENDS>
```

Appendix C. YANG identifiers as additional leaf

The device object is modelled as a container composed of an identifier and a list of instances. The list instance is composed of an instance number and a set of resource containers. The resource container is composed of the pair (attribute identifier number, IPSO resource specification).

```
<CODE BEGINS> file "ietf-humidityLF@2016-07-25.yang"

module ietf-humidityLF{

  yang-version 1.1; // needed for rpc

  namespace
    "urn:ietf:params:xml:ns:yang:ietf-humidityLF";

  prefix humid;

  organization
    "IPSO";

  contact
    "WG Web:  http://tools.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:   Peter van der Stok
    mailto:consultancy@vanderstok.org

    Editor:   Jaime Jimenez
    mailto:jaime.jimenez@ericsson.com";
```

```
description
  "This module contains information about the
operation of the IPSO LWM2M humidity sensor with ID 3304.
```

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.";

```
revision "2016-07-25" {
  description "Initial revision.";
  reference
    "I-D:draft-vanderstok-core-yang-lwm2m:
YANG language applied to the LWM2M IPSO humidity sensor
specification";
}

container IPSO-humidity{
  description
    "Device separated in identifier and list";
  leaf identifier{
    type uint16; // fixed to 3304
    config false;
    mandatory "true";
    description
      "the LWM2M identification number of the device";
  }
  list resources {
    key instance;
    config false; // should be same as key leaf
    description
      "3304: The humidity sensor is composed of
a set of instances";
    leaf instance {
      type uint16{
        range "0..1"; // only one instance zero (0)
      }
      config false; // R access
      mandatory "true";
      description
```

```
"the number of the humidity sensor instance";
} // instance number
container Sensor_Value {
  description
  "Resource separated in identifier and content";
  leaf identifier{
    type uint16; // fixed to 5700
    config false; // R access
    description
    "identifier should contain the value 5700";
  }
  leaf content{
    type decimal64{ // YANG has no float
      fraction-digits 2;
      range "10.0 .. 66.6";
    }
    units "Defined by 'Units' resource";
    config false; // R access
    mandatory "true";
    description
    "5700: Last or Current Measured Value
      from the Sensor";
  } // content
} // container
container Units {
  description
  "Resource separated in identifier and content";
  leaf identifier{
    type uint16; // fixed to 5701
    config false; // R access
    description
    "identifier should contain the value 5701";
  }
  leaf content{
    type string;
    units "Defined by 'Units' resource";
    config false; // R access
    description
    "5701: Measurement unit definition
      e.g. 'Cel' for temperature in Celsius";
  } // content
} // container
container Min_Measured_Value {
  description
  "Resource separated in identifier and content";
  leaf identifier{
    type uint16; // fixed to 5601
    config false; // R access
    description
```

```
"identifier should contain the value 5601";
} // identifier
leaf content{
type decimal64{ // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false; // R access
description
"5601: The minimum value measured by the
sensor since power ON or reset";
} // content
}
container Max_Measured_Value {
description
"Resource separated in identifier and content";
leaf identifier{
type uint16; // fixed to 5602
config false; // R access
description
"identifier should contain the value 5602";
}
leaf content{
type decimal64{ // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false; // R access
description
"5602: The maximum value measured by
the sensor since power ON or reset";
} // content
} // container
container Min_Range_Value {
description
"Resource separated in identifier and content";
leaf identifier{
type uint16; // fixed to 5603
config false; // R access
description
"identifier should contain the value 5603";
} // identifier
leaf content{
type decimal64{ // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false; // R access
```

```
description
"5603: The minimum value that can be measured
  by the sensor";
} // content
} // container
container Max_Range_Value{
description
"Resource separated in identifier and content";
leaf identifier{
type uint16;    // fixed to 5604
config false;  // R access
description
"identifier should contain the value 5604";
} // identifier
leaf content{
type decimal64{    // YANG has no float
fraction-digits 2;
range "10.0 .. 66.6";}
units "Defined by 'Units' resource";
config false;    // R access
description
"5604: The maximum value that can be measured
  by the sensor";
} // content
}
container Reset_Min_and_Max_measured_values {
description
"Resource separated in identifier and action";
leaf identifier{
type uint16;    // fixed to 5605
config false;  // R access
description
"identifier should contain the value 5605";
}
action reset{
// E access: this is an RPC without input and output parameters
description
"5605: Reset the Min and Max measured values to
  current value";
} // action reset
} // container Reset_min_and_max
} // list resources
} // container IPSO-humidity (3304)
} // module ietf-yang-humidity

<CODE ENDS>
```


Authors' Addresses

Peter van der Stok (editor)
consultant

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

Jaime Jimenez
Ericsson
Hirsalantie 11
Jorvas 02420
Finland

Phone: +358-442992827 (Finland)
Email: jaime.jimenez@ericsson.com

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2020

M. Veillette, Ed.
Trilliant Networks Inc.
I. Petrov, Ed.
Acklio
July 08, 2019

Constrained YANG Module Library
draft-veillette-core-yang-library-05

Abstract

This document describes a constrained version of the YANG library that provides information about the YANG modules, datastores, and datastore schemas used by a constrained network management server (e.g., a CORECONF server).

Status of This Memo

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

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

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

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

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

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

Table of Contents

1. Introduction	2
2. Terminology and Notation	2
3. Overview	3
3.1. Tree diagram	3
3.2. Major differences between ietf-constrained-yang-library and ietf-yang-library	4
4. YANG Module "ietf-constrained-yang-library"	5
5. IANA Considerations	13
5.1. YANG Module Registry	13
6. Security Considerations	13
7. Acknowledgments	14
8. References	14
8.1. Normative References	14
8.2. Informative References	14
Authors' Addresses	15

1. Introduction

There is a need for a standard mechanism to expose which YANG modules, datastores and datastore schemas are in use by a constrained network management server. This document defines the YANG module 'ietf-constrained-yang-library' that provides this information.

YANG module 'ietf-constrained-yang-library' shares the same data model and objectives as 'ietf-yang-library', only datatypes and mandatory requirements have been updated to minimize its size to allow its implementation by Constrained Nodes and/or Constrained Networks as defined by [RFC7228]. To review the list of objectives and proposed data model, please refer to [RFC8525] section 2 and 3.

2. Terminology and Notation

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

The following terms are defined in [RFC7950]: client, deviation, feature, module, submodule and server.

The following term is defined in [I-D.ietf-core-sid]: YANG Schema Item Identifier (SID).

The following terms are defined in [RFC8525]: YANG library and YANG library checksum.

3. Overview

The conceptual model of the YANG library is depicted in Figure 1.

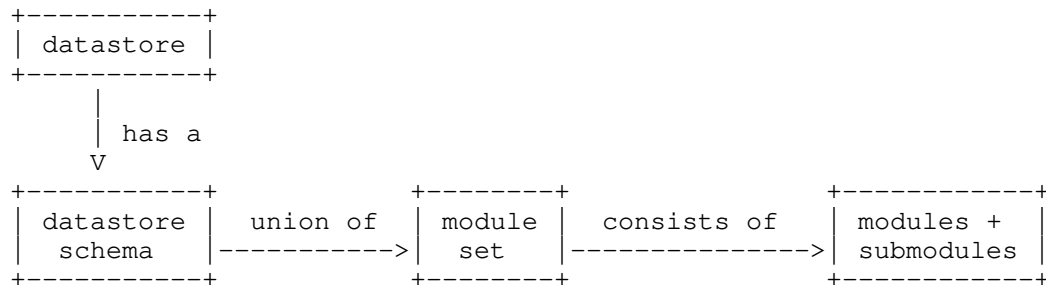


Figure 1: Conceptual model of the YANG library

It's expected that most constrained network management servers have one datastore (e.g. a unified datastore). However, some servers may have multiples datastore as described by NMDA [RFC8342]. The YANG library data model supports both cases.

In this model, every datastore has an associated datastore schema, which is the union of module sets, which is a collection of modules. Multiple datastores may refer to the same datastore schema and individual datastore schemas may share module sets.

For each module, the YANG library provides:

- o the YANG module identifier (i.e. SID)
- o its revision
- o its list of submodules
- o its list of imported modules
- o its set of features and deviations

YANG module namespace and location are also supported, but their implementation is not recommended for constrained servers.

3.1. Tree diagram

The tree diagram of YANG module `ietf-constrained-yang-library` is provided below. This graphical representation of a YANG module is defined in [RFC8340].

```

module: ietf-constrained-yang-library
  +--ro yang-library
    +--ro module-set* [index]
      +--ro index          uint8
      +--ro module* [identifier]
        +--ro identifier    sid:sid
        +--ro revision?     revision-identifier
        +--ro namespace?    inet:uri
        +--ro location*     inet:uri
        +--ro submodule* [identifier]
          +--ro identifier    sid:sid
          +--ro revision?     revision-identifier
          +--ro location*     inet:uri
        +--ro feature*      sid:sid
        +--ro deviation*    -> ../../module/identifier
      +--ro import-only-module* [identifier revision]
        +--ro identifier    sid:sid
        +--ro revision      union
        +--ro namespace?    inet:uri
        +--ro location*     inet:uri
        +--ro submodule* [identifier]
          +--ro identifier    sid:sid
          +--ro revision?     revision-identifier
          +--ro location*     inet:uri
    +--ro schema* [index]
      +--ro index          uint8
      +--ro module-set*    -> ../../module-set/index
    +--ro datastore* [identifier]
      +--ro identifier      ds:datastore-ref
      +--ro schema          -> ../../schema/index
    +--ro checksum          binary

notifications:
  +---n yang-library-update
    +--ro checksum          -> /yang-library/checksum

```

3.2. Major differences between ietf-constrained-yang-library and ietf-yang-library

The changes between the reference data model 'ietf-yang-library' and its constrained version 'ietf-constrained-yang-library' are listed below:

- o module-set 'name' and schema 'name' are implemented using an 8 bits unsigned integer and renamed 'index'.

- o module 'name', submodule 'name' and datastore 'name' are implemented using a SID (i.e. an unsigned integer) and renamed 'identifier'.
- o 'feature' and 'deviation' are implemented using a SID (i.e. an unsigned integer).
- o 'revision' fields are implemented using a 4 bytes binary string.
- o the mandatory requirement of the 'namespace' fields is removed, and implementation is not recommended. SIDs used by constrained devices and protocols don't require namespaces.
- o the implementation of the 'location' fields are not recommended, the use of the module SID as the handle to retrieve the associated YANG module is proposed instead.

4. YANG Module "ietf-constrained-yang-library"

RFC Ed.: update the date below with the date of RFC publication and remove this note.

```
<CODE BEGINS> file "ietf-constrained-yang-library@2019-03-28.yang"
module ietf-constrained-yang-library {
  yang-version 1.1;
  namespace
    "urn:ietf:params:xml:ns:yang:ietf-constrained-yang-library";
  prefix "yanglib";

  // RFC Ed.: update ietf-core-sid reference.

  import ietf-sid-file {
    prefix sid;
    reference "I-D.ietf-core-sid";
  }
  import ietf-inet-types {
    prefix inet;
    reference "RFC 6991: Common YANG Data Types.";
  }
  import ietf-datastores {
    prefix ds;
    reference
      "RFC 8342: Network Management Datastore Architecture (NMDA).";
  }

  organization
    "IETF NETCONF (Network Configuration) Working Group";
```

contact

"WG Web: <<http://datatracker.ietf.org/wg/core/>>
WG List: <<mailto:core@ietf.org>>
WG Chair: Carsten Bormann
<<mailto:cabo@tzi.org>>
WG Chair: Jaime Jimenez
<<mailto:jaime.jimenez@ericsson.com>>
Editor: Michel Veillette
<<mailto:michel.veillette@trilliantinc.com>>";

description

"This module provides information about the YANG modules, datastores, and datastore schemas implemented by a constrained network management server.

Copyright (c) 2018 IETF Trust and the persons identified as authors of the code. All rights reserved.

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

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

// RFC Ed.: update reference.

```
revision 2019-03-28 {  
  description  
    "Second revision.";  
  reference  
    "[I-D.veillette-core-yang-library]";  
}
```

```
revision 2018-09-21 {  
  description  
    "Initial revision.";  
  reference  
    "[I-D.veillette-core-yang-library]";  
}
```

```
/*
 * Typedefs
 */

typedef revision-identifier {
  type binary {
    length "4";
  }
  description
    "Revision date encoded as a binary string, each nibble
    representing a digit of the of revision date. For example,
    revision 2018-09-21 is encoded as 0x20 0x18 0x09 0x21.";
}

/*
 * Groupings
 */

grouping module-identification-leafs {
  description
    "Parameters for identifying YANG modules and submodules.";

  leaf identifier {
    type sid:sid;
    mandatory true;
    description
      "SID assigned to this module or submodule.";
  }
  leaf revision {
    type revision-identifier;
    description
      "The YANG module or submodule revision date. If no
      revision statement is present in the YANG module
      or submodule, this leaf is not instantiated.";
  }
}

grouping location-leaf-list {
  description
    "Common location leaf list parameter for modules and
    submodules.";

  leaf-list location {
    type inet:uri;
    description
      "Contains a URL that represents the YANG schema resource
      for this module or submodule."
  }
}
```



```
        This leaf is present in the model to keep the alignment
        with 'ietf-yang-library'. Support of this leaf in
        constrained devices is not necessarily required, nor
        expected. It is recommended that clients used the module
        or sub-module SID as the handle used to retrieve the
        corresponding YANG module";
    }
}

grouping implementation-parameters {
  description
    "Parameters for describing the implementation of a module.";

  leaf-list feature {
    type sid:sid;
    description
      "List of all YANG feature names from this module that are
      supported by the server, regardless whether they are
      defined in the module or any included submodule.";
  }
  leaf-list deviation {
    type leafref {
      path "../..../module/identifier";
    }
    description
      "List of all YANG deviation modules used by this server to
      modify the conformance of the module associated with this
      entry. Note that the same module can be used for
      deviations for multiple modules, so the same entry MAY
      appear within multiple 'module' entries.

      This reference MUST NOT (directly or indirectly)
      refer to the module being deviated.

      Robust clients may want to make sure that they handle a
      situation where a module deviates itself (directly or
      indirectly) gracefully.";
  }
}

grouping module-set-parameters {
  description
    "A set of parameters that describe a module set.";

  leaf index {
    type uint8;
    description
      "An arbitrary number assigned of the module set.";
  }
}
```

```
}
list module {
  key "identifier";
  description
    "An entry in this list represents a module implemented
    by the server, as per RFC 7950 section 5.6.5, with a
    particular set of supported features and deviations.";
  reference
    "RFC 7950: The YANG 1.1 Data Modeling Language.";

  uses module-identification-leafs;

  leaf namespace {
    type inet:uri;
    description
      "The XML namespace identifier for this module.
      This leaf is present in the model to keep the alignment
      with 'ietf-yang-library'. Support of this leaf in
      constrained devices is not required, nor expected.";
  }

  uses location-leaf-list;

  list submodule {
    key "identifier";
    description
      "Each entry represents one submodule within the parent
      module.";
    uses module-identification-leafs;
    uses location-leaf-list;
  }

  uses implementation-parameters;
}
list import-only-module {
  key "identifier revision";
  description
    "An entry in this list indicates that the server imports
    reusable definitions from the specified revision of the
    module, but does not implement any protocol accessible
    objects from this revision.

    Multiple entries for the same module name MAY exist.
    This can occur if multiple modules import the same
    module, but specify different revision-dates in the
    import statements.";

  leaf identifier {
```

```
    type sid:sid;
    description
      "The YANG module name.";
  }
  leaf revision {
    type union {
      type revision-identifier;
      type string {
        length 0;
      }
    }
    description
      "The YANG module revision date.";
  }
  leaf namespace {
    type inet:uri;
    description
      "The XML namespace identifier for this module.
      This leaf is present in the model to keep the alignment
      with 'ietf-yang-library'. Support of this leaf in
      constrained devices is not required, nor expected.";
  }

  uses location-leaf-list;

  list submodule {
    key "identifier";
    description
      "Each entry represents one submodule within the
      parent module.";

    uses module-identification-leafs;
    uses location-leaf-list;
  }
}

grouping yang-library-parameters {
  description
    "The YANG library data structure is represented as a grouping
    so it can be reused in configuration or another monitoring
    data structure.";

  list module-set {
    key index;
    description
      "A set of modules that may be used by one or more schemas.
```

```

    A module set does not have to be referentially complete,
    i.e., it may define modules that contain import statements
    for other modules not included in the module set.";

    uses module-set-parameters;
}

list schema {
  key "index";
  description
    "A datastore schema that may be used by one or more
    datastores.

    The schema must be valid and referentially complete,
    i.e., it must contain modules to satisfy all used import
    statements for all modules specified in the schema.";

  leaf index {
    type uint8;
    description
      "An arbitrary reference number assigned to the schema.";
  }
  leaf-list module-set {
    type leafref {
      path "../../module-set/index";
    }
    description
      "A set of module-sets that are included in this schema.
      If a non import-only module appears in multiple module
      sets, then the module revision and the associated
      features and deviations must be identical.";
  }
}

list datastore {
  key "identifier";
  description
    "A datastore supported by this server.

    Each datastore indicates which schema it supports.

    The server MUST instantiate one entry in this list
    per specific datastore it supports.

    Each datastore entry with the same datastore schema
    SHOULD reference the same schema.";

  leaf identifier {
```

```
        type ds:datastore-ref;
        description
            "The identity of the datastore.";
    }
    leaf schema {
        type leafref {
            path "../../schema/index";
        }
        mandatory true;
        description
            "A reference to the schema supported by this datastore.
            All non import-only modules of the schema are
            implemented with their associated features and
            deviations.";
    }
}

/*
 * Top-level container
 */

container yang-library {
    config false;
    description
        "Container holding the entire YANG library of this server.";

    uses yang-library-parameters;

    leaf checksum {
        type binary;
        mandatory true;
        description
            "A server-generated checksum or digest of the contents of
            the 'yang-library' tree. The server MUST change the
            value of this leaf if the information represented by
            the 'yang-library' tree, except 'yang-library/checksum',
            has changed.";
    }
}

/*
 * Notifications
 */

notification yang-library-update {
    description
        "Generated when any YANG library information on the
```

```
        server has changed.";

    leaf checksum {
        type leafref {
            path "/yanglib:yang-library/yanglib:checksum";
        }
        mandatory true;
        description
            "Contains the YANG library checksum or digest for the
             updated YANG library at the time the notification is
             generated.";
    }
}
}
}
<CODE ENDS>
```

5. IANA Considerations

5.1. YANG Module Registry

This document registers one YANG module in the YANG Module Names registry [RFC7950].

name: ietf-constrained-yang-library

namespace: urn:ietf:params:xml:ns:yang:ietf-constrained-yang-library

prefix: lib

reference: RFC XXXX

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

6. Security Considerations

Some of the readable data nodes in this YANG module may be considered sensitive or vulnerable in some network environments. It is thus important to control read access to these data nodes.

Specifically, the 'module' list may help an attacker to identify the server capabilities and server implementations with known bugs. Server vulnerabilities may be specific to particular modules, module revisions, module features, or even module deviations. This information is included in each module entry. For example, if a particular operation on a particular data node is known to cause a server to crash or significantly degrade device performance, then the module list information will help an attacker to identify server

implementations with such a defect, in order to launch a denial of service attack on these devices.

7. Acknowledgments

The YANG module defined by this memo have been derived from an already existing YANG module, `ietf-yang-library` [RFC8525], we will like to thanks to the authors of this YANG module. A special thank also to Andy Bierman for his initial recommendations for the creation of this YANG module.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <<https://www.rfc-editor.org/info/rfc7950>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8340] Bjorklund, M. and L. Berger, Ed., "YANG Tree Diagrams", BCP 215, RFC 8340, DOI 10.17487/RFC8340, March 2018, <<https://www.rfc-editor.org/info/rfc8340>>.
- [RFC8342] Bjorklund, M., Schoenwaelder, J., Shafer, P., Watsen, K., and R. Wilton, "Network Management Datastore Architecture (NMDA)", RFC 8342, DOI 10.17487/RFC8342, March 2018, <<https://www.rfc-editor.org/info/rfc8342>>.
- [RFC8525] Bierman, A., Bjorklund, M., Schoenwaelder, J., Watsen, K., and R. Wilton, "YANG Library", RFC 8525, DOI 10.17487/RFC8525, March 2019, <<https://www.rfc-editor.org/info/rfc8525>>.

8.2. Informative References

- [I-D.ietf-core-sid] Veillette, M., Pelov, A., and I. Petrov, "YANG Schema Item Identifier (SID)", draft-ietf-core-sid-07 (work in progress), July 2019.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <<https://www.rfc-editor.org/info/rfc7228>>.

Authors' Addresses

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

Email: michel.veillette@trilliantinc.com

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

Email: ivaylo@ackl.io

CoRE
Internet Draft
Intended status: Standards Track
Expires: September 4, 2018

P. Wang
C. Pu
H. Wang
J. Wu
Y. Yang
L. Shao
Chongqing University of
Posts and Telecommunications
J. Hou
Huawei Technologies
March 03, 2018

OPC UA Message Transmission Method over CoAP
draft-wang-core-opcua-transmission-03

Abstract

OPC Unified Architecture (OPC UA) is a data exchange specification that provides interoperability in industrial automation. With the arrival of Industry 4.0, it is of great importance to implement the exchange of semantic information utilizing OPC UA Transmitting in CoAP. This document provides some transmission methods for message of OPC UA over CoAP.

Status of this Memo

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

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

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

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

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

This Internet-Draft will expire on September 4, 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	2
1.1. Conventions and Terminology	3
2. Overview of OPC UA	3
2.1. Protocol Stack	3
2.2. Request/Response Model	5
3. Specification of OPC UA over CoAP	6
4. Transmission scheme	7
4.1. Direct transmission	7
4.2. REST transmission for OPC UA	8
5. Publish subscription for OPC UA over CoAP	9
6. Use Cases of OPC UA over CoAP	9
6.1. Factory data monitoring based on web pages	9
6.2. Offline/Online diagnostic system for resource-constrained factories	10
6.3. Factory data analysis based on cloud	11
7. Security Considerations	11
8. IANA Considerations	11
9. References	11
9.1. Normative References	11
9.2. Informative References	12
Authors' Addresses	13

1. Introduction

Internet of things is one of the attractive applications for CoAP [RFC7252]. Utilizing OPC UA [IEC TR 62541-1] Transmitting over CoAP could meet the demand for industry 4.0 based on the exchange of semantic information [I-D.wang-core-opcua-transmission-requirements]. In resource-constrained scenarios, OPC UA can effectively use energy, improve productivity and shorten the product manufacturing cycle by

building information model and using its cross-platform characteristic. Similar to OPC UA, CoAP message is exchanged in server/client mode. However, both of them have specific clients and servers. Driven by this, to implement OPC UA Transmitting over CoAP, the main problem to be solved is how OPC UA packets are transmitted over CoAP. For the transport layer of OPC UA, the main message transmission method is TCP or HTTP. It is worth noting that the design of CoAP is inspired by HTTP, thus, there are some similarities in transmission method between them. This document provides some transmission methods for message of OPC UA over CoAP, so that the communication between OPC UA client and OPC UA server could be established.

1.1. Conventions and Terminology

The keywords "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].

OPC: OLE for Process Control

OPC UA: OPC Unified Architecture

SOAP: Simple Object Access Protocol

REST: Representational State Transfer

HMI: Human Machine Interface

2. Overview of OPC UA

OPC Unified Architecture (OPC UA), standardized as IEC 62541, is a client-server communication protocol developed by OPC Foundation for safety, reliable data exchange in industrial automation. It is the evolution product of OPC (OLE for Process Control, where OLE denotes Object Linking and Embedding), the widely used standard process for automation technology, and is of great importance in realizing industry 4.0. By introducing Service-oriented architecture (SOA), OPC UA enables an open, cross-platform communication with the advantages of web services, robust security and integrated data model.

2.1. Protocol Stack

OPC UA is an application layer protocol that can be built on existing layers 5, 6 or 7 protocols such as TCP/IP, TLS or HTTP. The

OPC UA application layer consists of four sublayers: UA Application, Serialization Layer, Secure Channel Layer and Transport Layer (see Figure 1).

Serialization Layer includes two kinds of data encoding methods: UA Binary and UA XML. The UA XML, based on SOAP/HTTP or SOAP/HTTPS, is firewall friendly. On the other hand, the UA Binary, with least overhead and resource cost, offers an optimized speed and throughput.

The security layer varies according to the selected encoding format. For the HTTPS-based situation, security is implemented at TLS but Security Channel should still be presented even empty. It is worthwhile noting that the communication based on SOAP/HTTP has been deprecated since 2015, due to the lack of industrial approbation in the WS Secure Conversation.

For the transport layer (not the layer in OSI 7 layer model), options can be UA TCP, HTTPS, SOAP/HTTPS, and SOAP/HTTP. OPC UA defines a UA TCP protocol, which differs from HTTP in two main features: the allowance of responses to be returned in any order and to be returned on a different TCP transport end-point. In addition, UA TCP defines the interaction with the upper security channel.

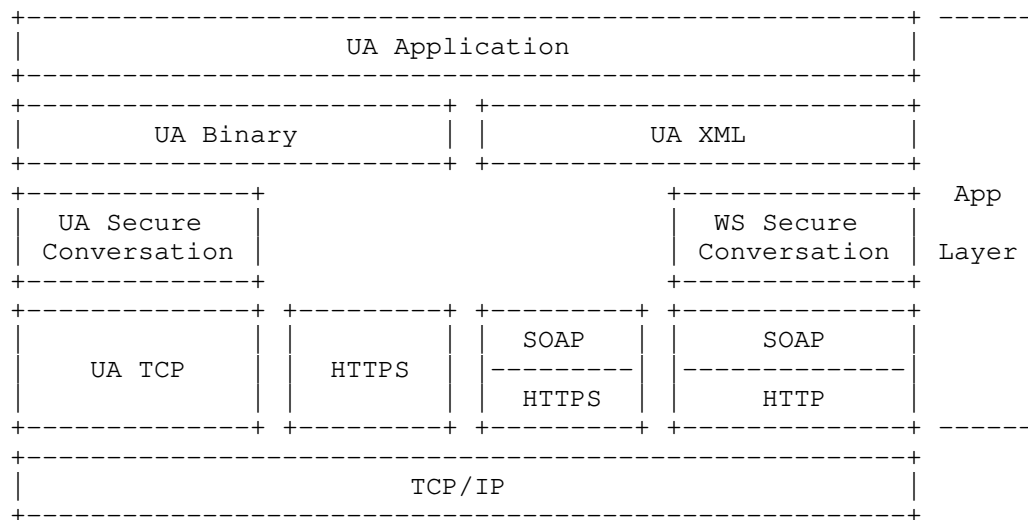


Figure 1: Layering of OPC UA over TCP/IP

2.2. Request/Response Model

The message exchange in UA binary mode is illustrated in Figure 2. After opening the socket, the client starts the connection with the server by using "hello" (HEL) and "acknowledge" (ACK) messages. Afterwards, a pair of messages is needed to open the security channel and define the encryption property. Then another two pairs of messages are exchanged so as to create and activate a session between the client and the server respectively. After these steps, the connection is initiated and the client can send request messages for services. When the request/response process is finished, a reverse process is required for disconnection.

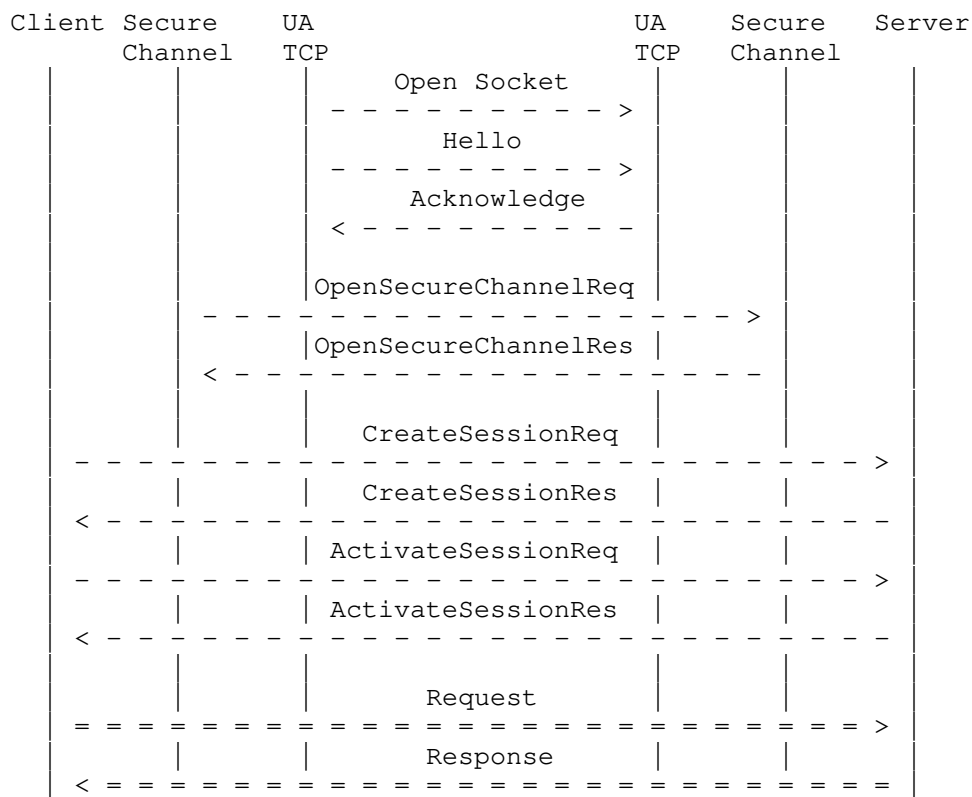


Figure 2: Request/Response Process of UA TCP

3. Specification of OPC UA over CoAP

As mentioned in section 2.1, OPC UA communications can be conducted through four options, among which two are related to HTTPS: HTTPS => UA Binary; HTTPS => SOAP => UA XML.

Constrained Application Protocol (CoAP) is an application layer protocol for constrained nodes and networks, which is designed to easily translate to HTTP for integration with the web. Although CoAP is built on the unreliable transport layer UDP, it offers a security mode binding to Datagram Transport Layer Security (DTLS). This document proposes a transmission scheme based on CoAPs (CoAP + DTLS) for constrained scenarios. The transmission based on CoAP over Transport Layer Security (TLS) is available [RFC8323].

The protocol stack of the CoAP based OPC UA is illustrated in Figure 3, including two options at Serialization Layer: UA Binary and UA XML. OPC UA packets are encoded in either binary or xml format, and the option field in the CoAP header can specify parameters that support both formats. Therefore, according to the format specified by the CoAP header, the entire packet of the OPC UA can be encapsulated in the payload of the CoAP message for direct transmission.

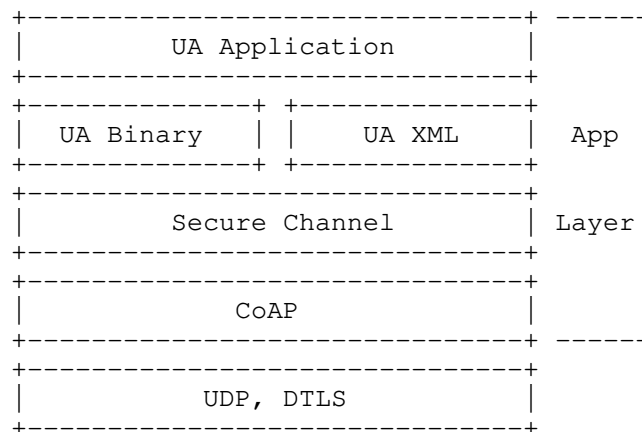


Figure 3: Layering of OPC UA over UDP

Both binary and XML encoding modes are based on the CoAP with an empty UA secure channel in between. For the XML encoding mode, since CoAP layer supports XML encoding format, the SOAP layer in the original stack is not needed.

4. Transmission scheme

4.1. Direct transmission

The transmission of OPC UA supports TCP protocol and HTTP protocol. CoAP is seen as a simplified HTTP protocol so that it can be applied to resource-constrained network. Therefore, this document considers the use of CoAP to directly transfer OPC UA messages. OPC UA packets are encoded in either binary or xml format, and the optional fields in the CoAP header specify parameters to support these two formats. Therefore, according to the format specified by the CoAP header, the entire packet of the OPC UA can be encapsulated in the payload of the CoAP message for direct transmission, as shown in Figure 4. According to CoAP, noted that this method of transmission needs to be modified on the server side and the client side of the OPC UA according to CoAP.

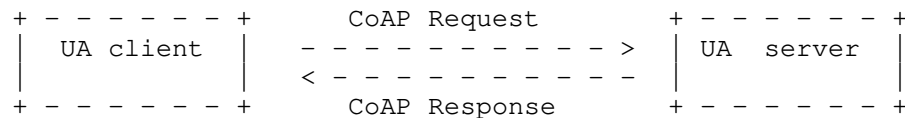


Figure 4: Direct transmission OPC UA based on CoAP

For supporting HTTP, a CoAP proxy can be established between OPC UA client and OPC UA server.

As shown in Figure 5, assuming all OPC UA servers are based on CoAP, and all OPC UA-CoAP servers can be considered to form a constrained network, then introducing a UA-to-CoAP proxy at the boundary of the network. When a traditional OPC UA client initiates an HTTP request to the UA-CoAP servers which is in the constrained network mentioned above, the UA-to-CoAP proxy maps the http request to the corresponding CoAP request and sends it to the UA-CoAP server in the network. After receiving the request, the UA-CoAP server sends a response to the UA-CoAP proxy. The proxy maps the CoAP response to the HTTP response and returns it to the UA client. For the UA client, the network proxy and conversion are transparent, in this way, the transfer of OPC UA in CoAP does not need to make any changes to the UA Client.

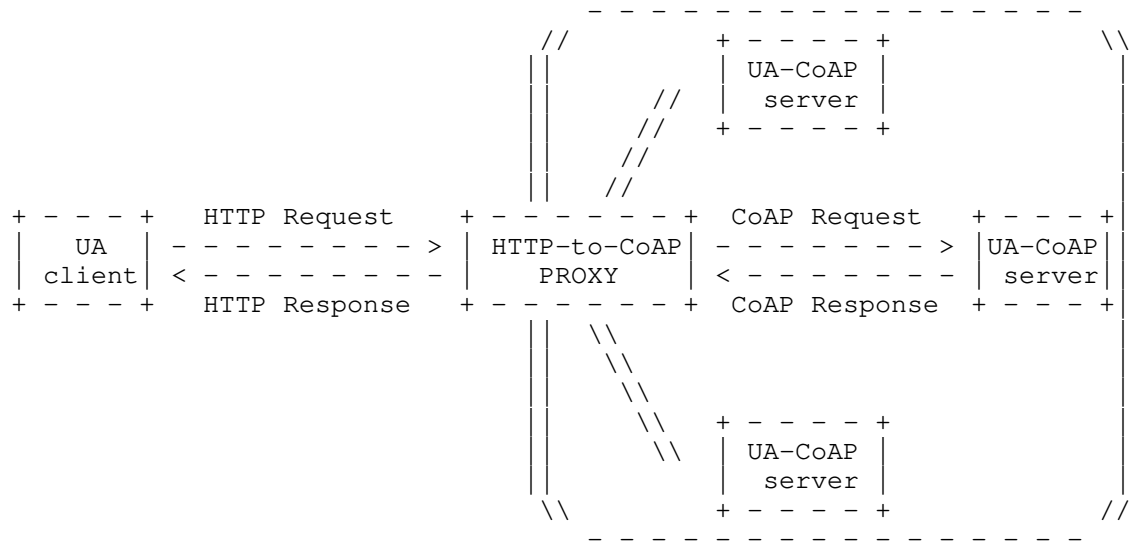


Figure 5: Proxy for OPC UA to CoAP

4.2. REST transmission for OPC UA

OPC UA is a set of data which exchange specifications for industrial communication, the core of the OPC UA protocol are information modeling and transmission, which marks each node in the address space with a unique identifier. A series of state interactions are needed before performing normal reading and writing, including message handshaking, opening a secure channel, creating a session, activating a session, etc. Besides, some states also need to be maintained during read and write operations.

In OPC UA, each node has an independent identifier in the address space, and different types of nodes can establish contact with each other by referencing. OPC UA defines a variety of services, and these services are fixed, because of this, the users cannot modify OPC UA services according to their own ideas. In general, services in OPC UA cannot be considered stateless, but many of them which are also commonly used are inherently stateless, e.g. FindServers, Read, Write [RICO]. The above features are in line with the REST architecture, due to CoAP is based on the REST architecture. Therefore, it is possible to simplify the interaction before the OPC UA performs the normal communication, and carry the OPC UA message by using the communication mode of the CoAP. Communication process is shown in Figure 6.

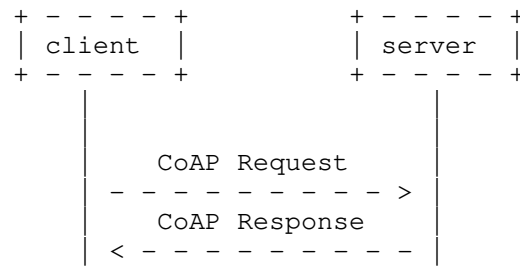


Figure 6: REST architecture communication of OPC UA

In Figure 2, the traditional OPC UA requires a series of interactions between normal read and write operations. Figure 6 shows that when using CoAP to carry OPC UA message, the interaction process is significantly reduced, which is conducive to the application of OPC UA in the restricted scenes. The cost of simplifying the interaction process is that the secure channel number is set to 0 by default, how to conduct secure data interaction needs further discussion.

5. Publish subscription for OPC UA over CoAP

As an application sublayer, CoAP provides publish-subscribe functionality, primarily for resource or network-constrained scenarios. Introducing proxy into the network [I-D.ietf-core-coap-pubsub], when a node needs to sleep, the node information is sent to the proxy agent, when another node requests to obtain information of this node, the broker release function can provide information. OPC UA defines the publish-and-subscribe function as a service in the service set. The client initiates the subscription request directly to the server, and the server periodically sends the information to the client. Comparing the characteristics of the two protocols, it is found that each of them has its own advantages. Joint design can be conducted for constrained applications.

TODO.

6. Use Cases of OPC UA over CoAP

6.1. Factory data monitoring based on web pages

Description: At present, the monitoring and management systems of the factory mostly exist in the form of software on the PC and the mobile. The drawback is that when the whole factory system is needed to upgrade, the monitoring software must be upgraded as well. It may cause the huge workload that will bring the time and financial

burden on the factory. CoAP is a HTTP-like communication protocol designed specifically for resource-constrained environments so that can be used in the factory because the sensor nodes in the factory mostly are resource-constrained. CoAP can easily transform to HTTP and OPC UA can consolidate the different protocols in the plant by building a unified information model.

Goal: PC and mobile devices can check and monitor the data by visiting WEB pages after CoAP is converted to HTTP. Avoiding large-scale software upgrades caused by system upgrades, while also reducing the development of mobile software, thereby reducing factory costs.

Requirements: the OPC UA information model should be encapsulated into CoAP data load. Because of the capacity limitation of UDP packet (MTU is 1472 bytes), in some cases, it is needed to compress, fragment, and reassemble packets.

6.2. Offline/Online diagnostic system for resource-constrained factories

Description: There are two modes existing in the factory's self-diagnosis system, the offline mode and the online mode. In the offline mode, the self-diagnostic device could use getHistorical, a service from OPC UA, to get historical Data. In the online mode, Both OPC UA and CoAP support pub/sub so that the monitoring system can obtain the data from a specific device in a short reaction time to determine its operating status. CoAP, as a resource-constrained factory transmission protocol, can easily access many web services APIs, add functionality that the factory can implement and let the system have a certain degree of expansibility. OPC UA could create a unified information model that realizes factory interoperability and protocol uniformity.

At same time, the controller node can diagnose and regulate other nodes by receiving their data rather than transferring them to HMI (The M2M Communication). Generally, using UDP is the best choice, however, CoAP's UDP not only has excellent stability but also has relatively few packet loss rates. The unified model of OPC UA enables all nodes to communicate without obstacles.

Goal: Using OPC UA over CoAP to enable factory offline history data diagnostics, online real-time monitoring, publish subscriptions and Achieving network nodes M2M communication.

Requirements: OPC UA uses SOA architecture, while CoAP uses REST architecture, it is necessary to design a reasonable architecture for OPC UA over CoAP.

6.3. Factory data analysis based on cloud

Description: Currently, there are many clouds (AWS, Windows Azure, etc.) which have different kinds of APIs. These clouds could achieve machine learning, data-flow analysis and so on for factory's data. Using CoAP can effectively access these interfaces and fully take advantage of clouds capabilities. At present, many factories have begun to use the cloud to improve production status, So the biggest benefit to use CoAP in factories is that CoAP could let devices to use cloud's applications in resource-constrained factories so that to achieve intelligent control. OPC UA can consolidate the different protocols in the plant by building a unified information model. Based on the content mentioned above, the field devices in the factories can transfer their data directly and immediately to the cloud without sending them to border routers or HMI.

Goal: Using OPC UA over CoAP to transfer field devices' data to the cloud.

Requirements: Using OPC UA to modeling the different types of data in the plant and then using CoAP to directly transfer the factory's data to the cloud.

7. Security Considerations

This document does not add any new security considerations beyond what the referenced technologies already have.

8. IANA Considerations

This memo includes no request to IANA.

9. References

9.1. Normative References

- [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol", RFC 7252, June 2014, <<https://tools.ietf.org/html/rfc7252>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997, <<https://tools.ietf.org/html/rfc2119>>.

- [RFC8075] Castellani, A., Loreto, S., and A. Rahman, "Guidelines for HTTP-to-CoAP Mapping Implementations", RFC 8075, November 2016, <<https://tools.ietf.org/html/rfc8075>>.
- [RFC8323] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", RFC8323, February 2018.

9.2. Informative References

- [IEC TR 62541-1]
IEC, "OPC unified architecture-Part1:Overview and concepts-IEC 62541", 2016, <https://webstore.iec.ch/preview/info_iec62541-1%7Bed2.0%7Den.pdf>.
- [I-D.wang-core-opcua-transmission-requirements]
Wang, H., Pu, C., Wang, P., Yang, Y., and D. Xiong, "Requirements Analysis for OPC UA over CoAP", draft-wang-core-opcua-transmission-requirements-02 (work in progress), December 2016.
- [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-03 (work in progress), February 2018.
- [RICO] Gruner, Sten, Julius Pfrommer, and Florian Palm. "RESTful industrial communication with OPC UA." IEEE Transactions on Industrial Informatics 12.5 (2016):1832-1841.
<<http://ieeexplore.ieee.org/abstract/document/7407396/>>

Authors' Addresses

Ping Wang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: wangping@cqupt.edu.cn

Chenggen Pu
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: mentospcg@163.com

Heng Wang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6248-7845
Email: wangheng@cqupt.edu.cn

Junrui Wu
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-18580183135
Email: wjr930914@gmail.com

Yi Yang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: 15023705316@163.com

Lun Shao
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: yjsslqcupt@163.com

Jianqiang Hou
Huawei Technologies CO.,LTD
101 Software Avenue,
Nanjing 210012
China

Phone: (86)-15852944235
Email: houjianqiang@huawei.com

Core
Internet Draft
Intended status: Standards Track
Expires: January 16, 2019

H. Wang
C. Pu
P. Wang
Y. Yang
D. Xiong
Chongqing University of
Posts and Telecommunications
July 15, 2018

Requirements Analysis for OPC UA over CoAP
draft-wang-core-opcua-transmission-requirements-03

Abstract

Constrained Application Protocol (CoAP) is an application protocol proposed for constrained nodes and constrained networks. Industrial Internet of Things (IIoT) is an attractive scenario for CoAP. OPC Unified Architecture (OPC UA) defines a semantic-based information model and a service-oriented architecture for IIoT, which can satisfy the requirements of Industry 4.0. This document analyses requirements for transmitting OPC UA over CoAP.

Status of this Memo

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

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

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

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

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

This Internet-Draft will expire on January 16, 2019.

Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<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. Architecture of OPC UA over CoAP.....	3
3. Requirements for OPC UA over CoAP.....	4
3.1. Encoding	4
3.2. Application Sublayer Optimization.....	4
3.3. Consistency	4
3.4. Reliability	5
3.5. Transmission Methods.....	5
3.6. Cache	5
3.7. Usability	5
4. Security Considerations.....	6
5. IANA Considerations	6
6. References	6
6.1. Normative References.....	6
6.2. Informative References.....	6
Authors' Addresses	8

1. Introduction

CoAP is a web application protocol designed for resource constrained devices and constrained networks which has been widely used in machine-to-machine (M2M) communications [RFC7252]. The purpose of applying CoAP to the Industrial Internet of Things (IIoT) is to provide connectivity for the devices. Whereas the communication of Industry 4.0 not only requires data value transmission, but also requires semantic information exchange. According to the definition of Industry 4.0 for communication, CoAP needs to support the exchange of semantic information, namely the semantic information model. For current protocols supporting semantic information model in the IIoT, the information model defined by OPC UA [IEC TR 62541-1] is very promising and its interactive model is similar to the

interactive model of CoAP, so it can be applied as a branch of the CoAP message payload.

2. Architecture of OPC UA over CoAP

To meet the needs of IIoT, the architecture of OPC UA over CoAP can be mainly divided into the following two patterns:

1) Figure 1 presents a logical layered structure of OPC UA Information Model over CoAP. In the transport layer, DTLS runs on top of UDP to secure transmission. Then, the middle layer utilizes the message mode defined in the CoAP protocol. Lastly, the information model of OPC UA [IEC TR 62541-5] is defined as an application of CoAP at the top. In such a hierarchical structure, the semantic-based data information in OPC UA can be transmitted in resources-constrained scenarios, so that CoAP can meet the requirements of semantic information transmission.

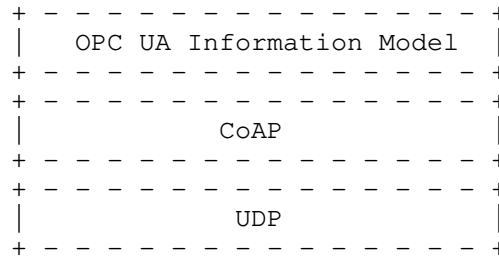


Figure 1: OPC UA Information Model over CoAP

2) In order to take full advantage of the service sets defined by OPC UA, this document proposes the other architecture for OPC UA

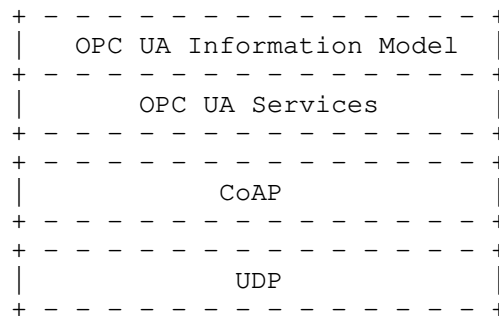


Figure 2: OPC UA Information Model and Services over CoAP

transmission over CoAP. As shown in Figure 2, the information model of OPC UA is defined as the application of CoAP, moreover, the connection establishment, creating session, publish/subscribe and other functions related to data information interaction are all implemented by the service sets defined by OPC UA. CoAP is mainly responsible for the definition of message format and runs over UDP to keep the implementation lightweight.

3. Requirements for OPC UA over CoAP

3.1. Encoding

CoAP messages are encoded in a simple binary format that starts with a fixed-size 4-byte header. The header is followed by a variable-length Token value, which can be between 0 and 8 bytes long. Following the Token value comes a sequence of zero or more CoAP Options in Type-Length-Value (TLV) format, optionally followed by a payload that takes up the rest of the datagram. In addition, the OPC UA protocol coding mainly includes two ways that are binary and XML. Therefore, in order to transmit the information model of OPC UA over CoAP, specific frame formats of CoAP need to be designed to support two kinds of coding modes of OPC UA.

3.2. Application Sublayer Optimization

For information exchange, the document [I-D.ietf-core-coap-pubsub] defines the corresponding application sublayer, OPC UA also defines a number of specific communication patterns. For example, in the new specification defined by OPC UA, there are two publish/subscribe modes. one is the Broker-less mode, another is Broker-based mode. Correspondingly, in the publish/subscribe specification of CoAP, it introduces broker mechanism in which the client sends the state information to the Broker and the Broker provides storage and forwarding function to implement the publish/subscribe function. Comparing above two protocols, they are achieved the publish/subscribe function by the Broker. But it is still necessary to optimize the application sublayer of CoAP to support some particular communication modes of OPC UA.

3.3. Consistency

The interactive model of CoAP is the client/server model. However, in M2M scenarios, CoAP entities often act as both servers and clients. Compared to OPC UA, though the interactive model is also the client/server model, there is a set of supported services in the OPC UA server. Consequently, for the great difference of the server definition of these two protocols, we need to tackle with the

consistency and integration issues between the CoAP server and the OPC UA server.

3.4. Reliability

One of the main design goals of CoAP is to satisfy some special requirements such as communication in the constrained scenarios that address power consumption. Hence, in order to reduce network overhead and avoid network congestion, CoAP is designed to run over UDP, which is a good choice to achieve inter-network data exchange in use of the IP architecture. However, UDP is a connectionless transport layer protocol that provides unreliable information transmission services. In the field of IIoT, we need to ensure the reliability of data transmission to avoid losing some important data information. Moreover, CoAP addresses transmission reliability by defining a message as requiring acknowledgment, obviously this is not enough to meet the high reliability requirements in the field of IIoT, so the reliability of COAP remains to be optimized.

3.5. Transmission Methods

For OPC UA over CoAP, one of the important issues that needs to be addressed is how to transmit messages. The connection between OPC UA client and server is stateful, the connection status need to be maintained in the process of message interaction, while CoAP is a stateless connection, so that the message transmission of the two protocols is different. Fortunately, the transport layer protocol of OPC UA supports TCP and HTTP, in addition, the CoAP protocol can be considered that it is improved for constrained scenarios based on HTTP. Therefore, a solution can be found for the messages transmission by using the similarity of two protocols in HTTP.

3.6. Cache

In order to reduce response time and network bandwidth consumption, CoAP provides caching responses in the endpoints. When the endpoint gets the request, it may use the old message to reply the request. It is meaningful for the resource-constrained devices to save resource. However, the information model of OPC UA does not support the mechanism that should be solved by proposing some ways.

3.7. Usability

For OPC UA over CoAP, it contains the key technologies of two different protocols. It is difficult for application developers to master the two protocols at the same time. Moreover, application developers usually focus on the implementation of the function, and do not care about the specific implementation process of the

underlying protocol. So, OPC UA over CoAP need to remain independent from the application. On the other hand, it should maintain the flexibility of configuration so that application developers can set it to satisfy different needs.

4. Security Considerations

The security of CoAP includes four modes in which three modes implemented based on the Datagram Transport Layer Security (DTLS) except the non-security mode. However, the security architecture of OPC UA is built on the application layer and the communication layer above the transport layer. Specifically, the application layer adopts the authentication and authorization, and the communication layer achieves the security of OPC UA [IEC TR 62541-2] through secure channel encryption. Though OPC UA has four modes, the security model of OPC UA is realized based on Transport Layer Security (TLS). Actually, DTLS is an addition to TLS to solve the unreliable transmission feature of UDP. Currently, some documents show that CoAP needs to support TLS. Therefore, the security of the two protocols can be implemented jointly.

5. IANA Considerations

This memo includes no request to IANA.

6. References

6.1. Normative References

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol", RFC 7252, June 2014, <<https://tools.ietf.org/html/rfc7252>>.

6.2. Informative References

[IEC TR 62541-1]
IEC, "OPC unified architecture-Part1: Overview and concepts-IEC 62541", 2016, <https://webstore.iec.ch/preview/info_iec62541-1%7Bed2.0%7Den.pdf>.

[IEC TR 62541-5]
IEC, "OPC unified architecture-Part5: Information Model-IEC 62541", 2015, <https://webstore.iec.ch/preview/info_iec62541-5%7Bed2.0%7Db.pdf>.

[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-02 (work in progress), July 2017.

[IEC TR 62541-2]

IEC, "OPC unified architecture-Part2: Security Model-IEC 62541", 2016, <
https://webstore.iec.ch/preview/info_iec62541-2%7Bed2.0%7Db.pdf>.

Authors' Addresses

Heng Wang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6248-7845
Email: wangheng@cqupt.edu.cn

Chenggen Pu
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: mentospcg@163.com

Ping Wang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: wangping@cqupt.edu.cn

Yi Yang
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: 15023705316@163.com

Daijing Xiong
Chongqing University of Posts and Telecommunications
2 Chongwen Road
Chongqing, 400065
China

Phone: (86)-23-6246-1061
Email: 15111825021@163.com

