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

Z. Cao
R. Jadhav
Huawei

October 27, 2016

CoAP Delegated Observe
draft-cao-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.

Table of Contents

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.

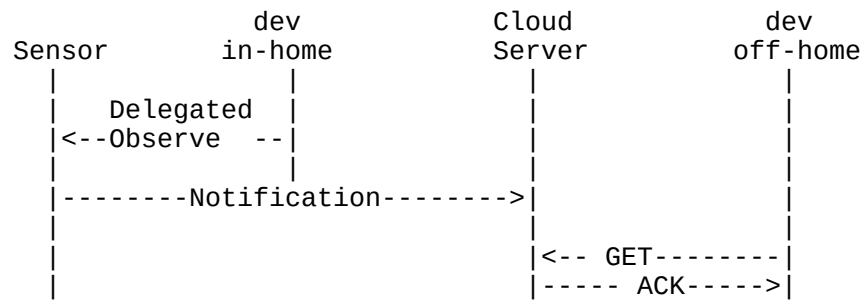
2.1 Multiple Devices

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

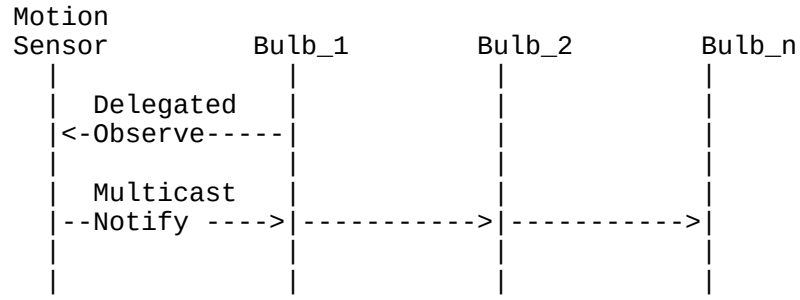
A user wants its mobile device to be notified of a certain sensor information both in-home and off-home. When the device moves out of its smart home network coverage, it is normally hidden behind NAT and FW that keep it being reached for such notification messages. In this case, the normal observe-notification 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.



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.



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.

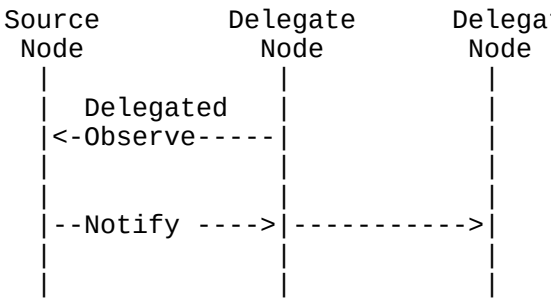


Figure 3: Overview of Delegated Observe

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

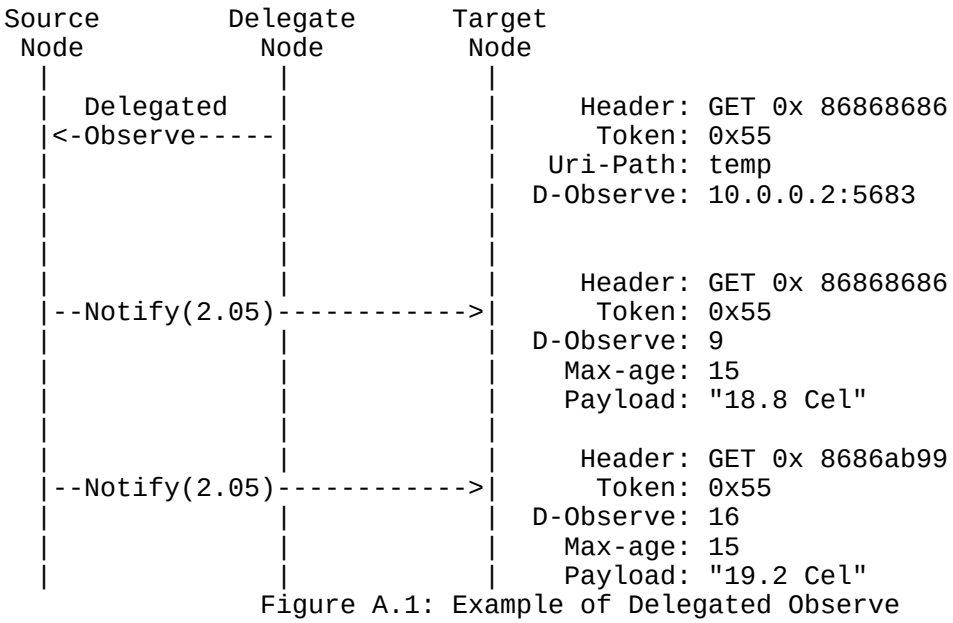


Figure A.1: Example of Delegated Observe

Authors' Addresses

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: 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-webrtcdc-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

Table of Contents

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.

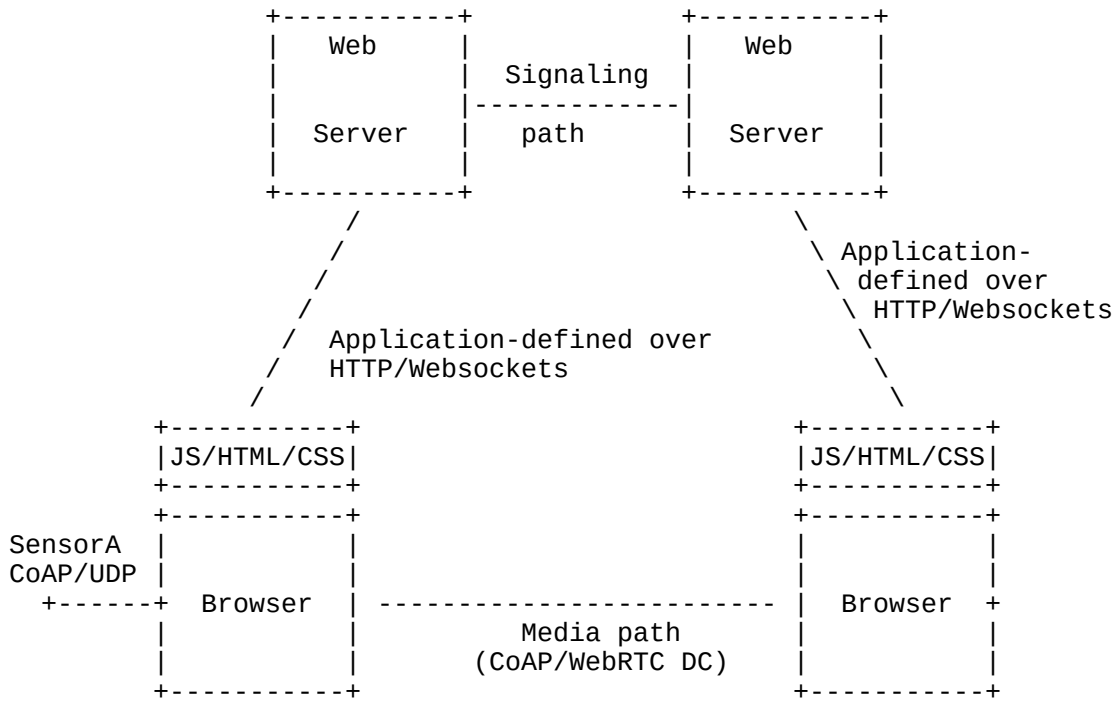


Figure 1: CoAP and WebRTC Trapeziod

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.

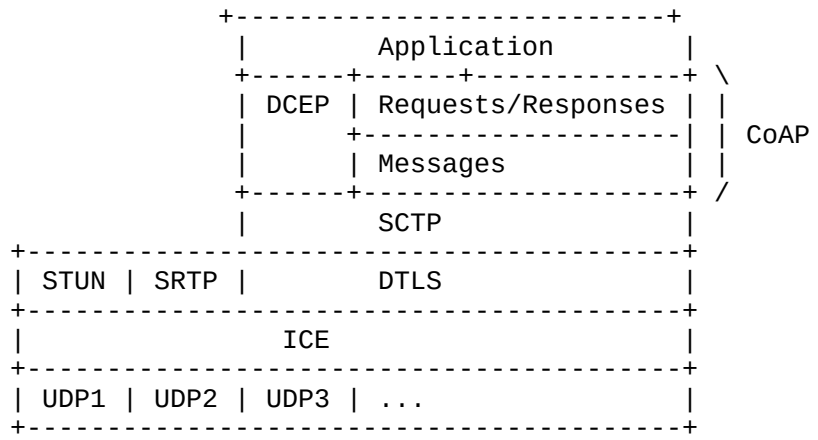


Figure 2: WebRTC protocol layers including CoAP

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:

- 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.

- 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.

- 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.

- an optional label: This is an application/implementation specific label. Uniqueness is not guaranteed. Usage of CoAP has no impact on this parameter.

- 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 intermediares 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 chracteristics. Consideration for the individual options are below:

- 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.

- Max-message size Capability: The CoAP Max-Message-Size shall not exceed the SCTP message size.

- 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.

- 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).

- 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.

- 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.

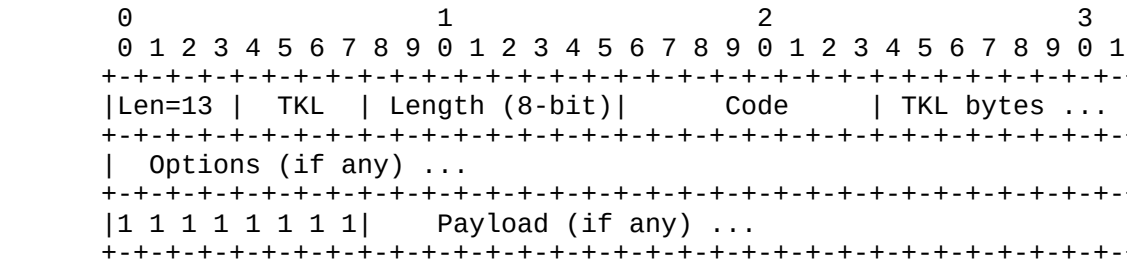


Figure 3: CoAP Header with TCP with 8-bit Length in Header

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

- 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:

- WebRTC String: to identify a non-empty JavaScript string encoded in UTF-8.

- 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:

- 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.
- section 5.3.1/[RFC7252]: due to the use of DTLS the advice regarding token use without using TLS is invalid.
- section 5.3.2/[RFC7252]: In addition CoAP request/response matching is unique to a particular WebRTC DC (SCTP StreamID pair).
- 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:

- 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:

- Map received Empty CoAP message to SCTP level operations and discard the empty message.
- Map received ACK message to SCTP level operations and discard the ACK message.
- Separate piggy-backed messages.
- 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:

- section 11.3/[RFC7252] on insecure UDP and multicast being used to aid an amplification attack.
- section 11.4/[RFC7252] on IP address spoofing and section 11.5/[RFC7252] on Cross-Protocol attacks.
- 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.

- Subprotocol Identifier: coap.v1

- Subprotocol Common Name: Constrained Application Protocol (CoAP)

- 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:

- CORE workgroup: core@ietf.org
- 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:ATEn1v9DoTMB9J4r
      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-webrtcdc-02:

- Keep alive update. No changes.

draft-groves-coap-webrtcdc-01:

- Updated message format to align with draft-core-coap-tcp-tls-04

- 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: 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

Table of Contents

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:

- 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.

- 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 reegistry.

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

- 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

Thing-to-Thing Research Group
Internet-Draft
Intended status: Experimental
Expires: January 9, 2020

K. Hartke
Ericsson

July 8, 2019

Thing-to-Thing Data Hub
draft-hartke-t2trg-data-hub-04

Abstract

The "Thing-to-Thing Data Hub" is a RESTful, hypermedia-driven Web application that can be used in Thing-to-Thing communications to share data items such as thing descriptions, configurations, resource descriptions, or firmware updates at a central location.

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.

Table of Contents

1. Introduction

In Thing-to-Thing communication, there is often a need to share data items of common interest through a central location. For example, the Resource Directory [I-D.ietf-core-resource-directory] aggregates descriptions of Web resources held on constrained nodes, which enables other nodes to easily discover these resources; a Thing Directory [W3C.CR-wot-architecture-20190516] stores metadata of IoT devices, allowing clients to discover interaction affordances and supported protocol bindings of Things; and a Firmware Server [I-D.ietf-suit-architecture] stores firmware images and manifests, making this data available to deployed devices, commissioning tools, and other services.

As more and more Thing-to-Thing applications are implemented, it becomes increasingly important being able to not only share resource descriptions and firmware updates but also many other kinds of data, such as default configurations for new devices, service locations, or certificate revocation lists. Resource directories and firmware servers are not a good fit for these kinds of data, as they're specialized to their use cases and generally not accepting any other kinds of data. And creating new, specialized applications for every type of data is not practical in the long term.

This document defines a simple "data hub" application, a RESTful Web application with a machine-understandable hypermedia API. A "data hub" generalizes the concept of a central repository for different applications and is suitable for constrained environments [RFC7228]. Specifically, it enables clients to share data items in any format and provides means for creating, reading, observing, updating, deleting, and finding data items at a data hub server.

Data hubs are intended to be used primarily with Constrained Application Protocol (CoAP) [RFC7252].

Features:

- General

  The data hub generalizes the concept of a directory or repository to data items of any Internet media type. This means that applications using the data hub aren't stuck forever with the same media types or limited to just resource descriptions or firmware updates.

- Searchable

  Clients can retrieve a subset of data items from a data hub based on item metadata.

- Observable

  Data items published to a data hub are exposed as resources. As such, they can be observed for changes [RFC7641] over CoAP. This allows clients to stay informed of information that other clients update over time. As a result, the data hub functions similar to a Publish-Subscribe Broker [I-D.ietf-core-coap-pubsub].

- Evolvable

  The key differentiator of the data hub compared to Resource Directory [I-D.ietf-core-resource-directory] and CoAP Publish-Subscribe Broker [I-D.ietf-core-coap-pubsub] lies in the evolvability of the application -- the ability to respond effectively to the need for changes without negatively impacting existing and new clients.

  Data hubs enable fine-grained evolvability by driving all interactions by machine-understandable hypermedia elements. Features can be added, changed or removed in a safe, backwards-compatible way simply by updating the data hub representation to expose appropriate links and forms.

1.1. Notational Conventions

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

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

2. Data Model

The data model of the "Thing-to-Thing Data Hub" application consists of three elements: the _data hub_ itself, a _data collection_, and a number of _data items_ that have been shared (Figure 1).

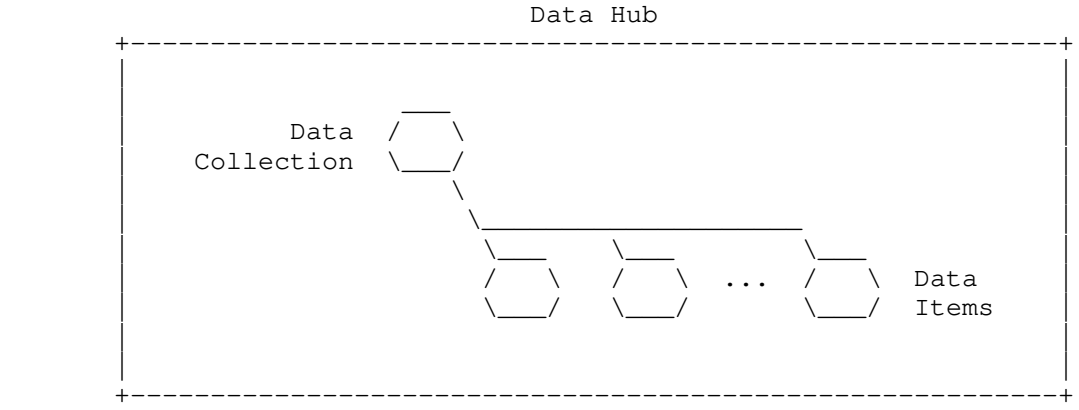


Figure 1: A Data Collection with a Number of Shared Data Items Hosted at a Data Hub

Data Hub

A data hub is a Web application running on a Web server that hosts the data collection and the data items.

Data Collection

A data collection is a collection resource that contains the data items.

Representations of data collections MUST have the "application/coral+cbor" media type [I-D.hartke-t2trg-coral]. The representations consist primarily of links to the data items. These links have the <http://www.iana.org/assignments/relation/item> link relation type [RFC6573]. To reduce the number of round-trips, the representations MAY also embed (complete or partial) representations of the data items. Forms contained in the representation enable interactions with the data collection and the data items, as detailed in the following sections. The representations MAY additionally contain other links and forms that are not described in this document, such as a link with the <http://www.iana.org/assignments/relation/alternate> link relation type [W3C.REC-html52-20171214] that references an alternate representation of the data collection resource.

In this version of this document, a data hub is defined to have a depth of only one level. That is, all data item resources are organized directly under the top-level data collection resource. This could be extended to multiple levels in a future version.

Data Item

A data item is a resource that is a member of the data collection resource.

Data item representations can have any media type. However, a data collection MAY restrict the media types it accepts for publication. In this case, the form in the representation of the data collection for creating data items MUST list the acceptable media types using form fields of type <http://coreapps.org/coap#accept>.

The representations of data items MAY link back to the data collection resource using the <http://www.iana.org/assignments/relation/collection> link relation type [RFC6573].

3. Interaction Model

The interaction model consists of eight potential interactions with a data hub: discovering and reading the data collection, and creating, reading, observing, updating, deleting, and finding shared data items in the data collection.

Discovering a Data Hub

In this version of this document, clients are assumed to be pre-configured with an entry-point IRI for a data collection at a data hub.

Reading a Collection

A client can retrieve a representation of a data collection by dereferencing the entry-point IRI. As described above, the representation of the data collection includes links to (and, optionally, representations of) the data items in the data collection. The representation of the data collection also includes forms for creating, updating, deleting, and finding data items.

Creating an Item

The representation of a data collection MAY contain a form with the <http://coreapps.org/collections#create> operation type. Submitting this form with a representation in one of the acceptable media types creates a new data item in the data collection. The acceptable media types are indicated by form fields of type <http://coreapps.org/coap#accept>.

Implementations of this version of this document MUST use the method implied by the <http://coreapps.org/collections#create> operation type, i.e., the POST method [RFC7252]. A form indicating different a method MUST be ignored.

On success, the location of the created data item MUST be conveyed in a 2.01 (Created) response using the Location-Path and Location-Query options [RFC7252].

Reading an Item

A client can retrieve a representation of a data item by following a link with the <http://www.iana.org/assignments/relation/item> link relation type in the representation of the data collection.

Observing an Item

A client can observe a data item by following a link with the <http://www.iana.org/assignments/relation/item> link relation type in the representation of the data collection and observing the target resource as specified in RFC 7641 [RFC7641].

Updating an Item

For each data item in a data collection, the representation of the data collection MAY include a form with the <http://coreapps.org/base#update> operation type nested within the link to the data item. Submitting this form updates the data item to the submitted representation.

Implementations of this version of this document MUST use the method implied by the <http://coreapps.org/base#update> operation type, i.e., the PUT method [RFC7252]. A form indicating different a method MUST be ignored.

On success, a 2.04 (Changed) response is returned.

Deleting an Item

For each data item in a data collection, the representation of the data collection MAY include a form with the <http://coreapps.org/

collections#delete> operation type nested within the link to the data item. Submitting this form deletes the data item from the data collection.

Implementations of this version of this document MUST use the method implied by the <http://coreapps.org/collections#delete> operation type, i.e., the DELETE method [RFC7252]. A form indicating different a method MUST be ignored.

On success, a 2.02 (Deleted) response is returned.

Searching for Items

The representation of a data collection MAY contain a form with the <http://coreapps.org/base#search> operation type. This form can be used to find data items in the data collection. Submitting this form with a search query returns the subset of data items that match the query.

(TODO: Specify the representation format for search queries.)

Implementations of this version of this document MUST use the method implied by the <http://coreapps.org/base#search> operation type, i.e., the FETCH method [RFC8132]. A form indicating different a method MUST be ignored.

On success, a 2.05 (Content) response is returned.

(TODO: Specify the representation format for the response.)

4. Security Considerations

The data hub application relies on a Web transfer protocol like CoAP to exchange representations in a CoRAL serialization format. See Section 11 of RFC 7252 [RFC7252] and Section 7 of RFC 7641 [RFC7641] for security considerations relating to CoAP. See Section 7 of RFC XXXX [I-D.hartke-t2trg-coral] for security considerations relating to CoRAL.

The data hub application does not define any specific mechanisms for protecting the confidentiality and integrity of messages exchanged between a data hub and a client. It is recommended that implementations employ application layer or transport layer mechanisms for interactions with a data hub.

The data hub application does not define any specific mechanisms for protecting the confidentiality and integrity of representations of data items shared through a data hub. For scenarios where end-to-end

security matters, such as for firmware updates [I-D.ietf-suit-information-model], implementations should employ an object security mechanism.

5. IANA Considerations

This document has no IANA actions.

6. References

6.1. Normative References

[I-D.hartke-t2trg-coral] Hartke, K., "The Constrained RESTful Application Language (CoRAL)", draft-hartke-t2trg-coral-09 (work in progress), July 2019.

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

[RFC6573] Amundsen, M., "The Item and Collection Link Relations", RFC 6573, DOI 10.17487/RFC6573, April 2012, <https://www.rfc-editor.org/info/rfc6573>.

[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>.

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-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-22 (work in progress), July 2019.

[I-D.ietf-suit-architecture]
Moran, B., Meriac, M., Tschofenig, H., and D. Brown, "A Firmware Update Architecture for Internet of Things Devices", draft-ietf-suit-architecture-05 (work in progress), April 2019.

[I-D.ietf-suit-information-model]
Moran, B., Tschofenig, H., and H. Birkholz, "Firmware Updates for Internet of Things Devices - An Information Model for Manifests", draft-ietf-suit-information-model-02 (work in progress), January 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>.

[W3C.CR-wot-architecture-20190516]
Kovatsch, M., Matsukura, R., Lagally, M., Kawaguchi, T., Toumura, K., and K. Kajimoto, "Web of Things (WoT) Architecture", World Wide Web Consortium Candidate Recommendation CR-wot-architecture-20190516, May 2019, <https://www.w3.org/TR/2019/CR-wot-architecture-20190516>.

[W3C.REC-html52-20171214]
Faulkner, S., Eicholz, A., Leithead, T., Danilo, A., and S. Moon, "HTML 5.2", World Wide Web Consortium Recommendation REC-html52-20171214, December 2017, <https://www.w3.org/TR/2017/REC-html52-20171214>.

Acknowledgements

Thanks to Christian Amsuess and Jaime Jimenez for helpful comments and discussions that have shaped the document.

Author's Address

Klaus Hartke
Ericsson
Torshamnsgatan 23
Stockholm SE-16483
Sweden

Email: klaus.hartke@ericsson.com

CORE
Internet-Draft
Updates: 7641 (if approved)
Intended status: Standards Track
Expires: April 14, 2017

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.
Microsoft

October 11, 2016

CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets
draft-ietf-core-coap-tcp-tls-05

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.

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 14, 2017.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] was designed for Internet of Things (IoT) deployments, assuming that UDP [RFC0768] or DTLS [RFC6347] over UDP can be used unimpeded. UDP is a good choice for transferring small amounts of data across networks that follow the IP architecture.

Some CoAP deployments need to integrate well with existing enterprise infrastructures, where UDP-based protocols may not be well-received or may even be blocked by firewalls. Middleboxes that are unaware of CoAP usage for IoT can make the use of UDP brittle, resulting in lost or malformed packets.

Emerging standards such as Lightweight Machine to Machine [LWM2M] currently use CoAP over UDP as a transport and require support for CoAP over TCP to address the issues above and to protect investments in existing CoAP implementations and deployments. Although HTTP/2 could also potentially address these requirements, there would be additional costs and delays introduced by such a transition. Currently, there are also fewer HTTP/2 implementations available for constrained devices in comparison to CoAP.

To address these requirements, this document defines how to transport CoAP over TCP, CoAP over TLS, and CoAP over WebSockets. Figure 1 illustrates the layering:

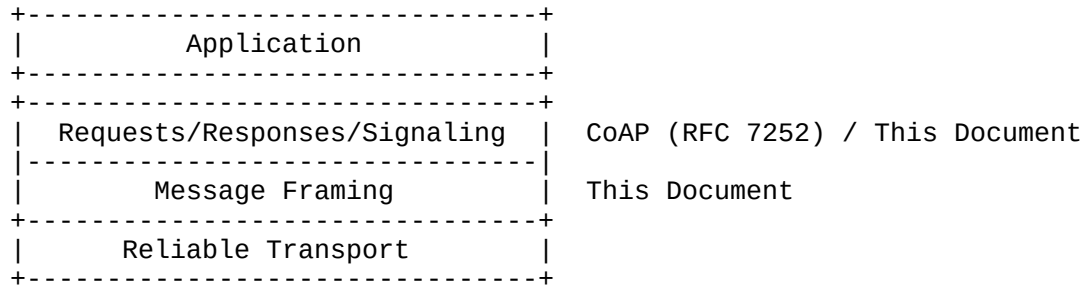


Figure 1: Layering of CoAP over Reliable Transports

Where NATs are present, CoAP over TCP can help with their traversal. 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 life cycle. UDP, on the other hand, does not provide such information to a NAT and timeouts tend to be much shorter [HomeGateway].

Some environments may also benefit from the ability of TCP to exchange larger payloads, such as firmware images, without application layer segmentation and to utilize the more sophisticated congestion control capabilities provided by many TCP implementations.

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.

To allow IoT devices to better communicate in these demanding environments, CoAP needs to support different transport protocols, namely TCP [RFC0793], in some situations secured by TLS [RFC5246].

In addition, some corporate networks only allow Internet access via a HTTP proxy. In this case, the best transport for CoAP would be the WebSocket Protocol [RFC6455]. The WebSocket protocol provides two-way communication between a client and a server after upgrading an HTTP/1.1 [RFC7230] connection and may be available in an environment that blocks CoAP over UDP. Another scenario for CoAP over WebSockets

is a CoAP application running inside a web browser without access to connectivity other than HTTP and WebSockets.

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.

1.1. Terminology

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

This document assumes that readers are familiar with the terms and concepts that are used in [RFC6455], [RFC7252], and [RFC7641].

The term "reliable transport" only refers to stream-based transport protocols such as TCP.

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 (Section 2.1 of [RFC7959]).

2. 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. CoAP over UDP supports optional reliability by defining four types of messages: Confirmable, Non-confirmable, Acknowledgement, and Reset. TCP eliminates the need for the message layer to support reliability. As a result, message types are not defined in CoAP over TCP.

2.1. Messaging Model

Conceptually, CoAP over TCP replaces most of the CoAP over UDP message layer 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 CoAP over TCP messaging layer is not required to support acknowledgements or detection of 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. All messages are also untyped.

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.

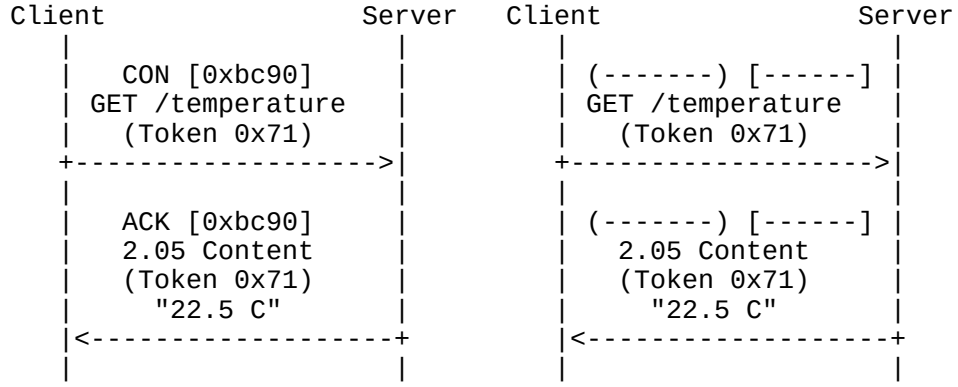


Figure 2: Comparison between CoAP over unreliable and reliable transport.

2.2. UDP-to-TCP gateways

A UDP-to-TCP gateway MUST discard all Empty messages (Code 0.00) after processing at the message layer. For Confirmable (CON), Non-Confirmable (NOM), and Acknowledgement (ACK) messages that are not Empty, their contents are repackaged into untyped messages.

2.3. Opening Handshake

Both the client and the server MUST send a Capability and Settings message (CSM see Section 4.3) as its 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 unnecessary latency, a client MAY send additional messages without waiting to receive the server CSM; however, it is important

to note that the server CSM might advertise capabilities that impact how a client is expected to communicate with the server. For example, the server CSM could advertise a Max-Message-Size option (See Section 4.3.2) that is smaller than the base value (1152).

Clients and servers MUST treat a missing or invalid CSM as a connection error and abort the connection (see Section 4.6).

2.4. 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.

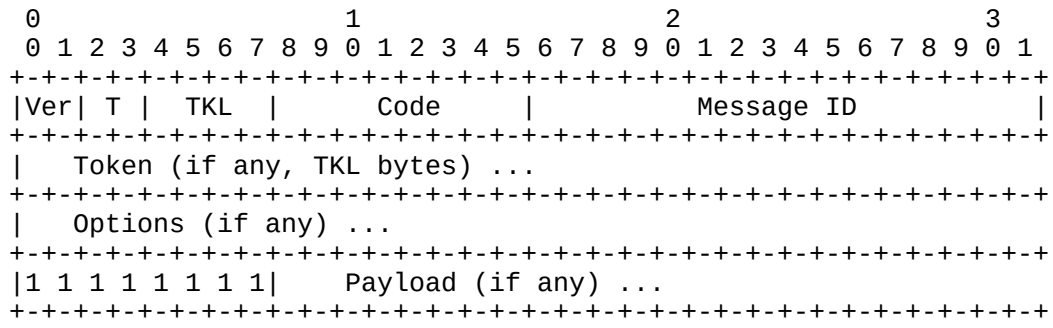


Figure 3: RFC 7252 defined CoAP Message Format.

The CoAP over TCP message format is very similar to the format specified for CoAP over UDP. The differences are as follows:

- Since the underlying TCP connection provides retransmissions and deduplication, there is no need for the reliability mechanisms provided by CoAP over UDP. The "T" and "Message ID" fields in the CoAP message header are elided.

- The "Ver" field is elided as well. In constrast to the UDP message layer for UDP and DTLS, the CoAP over TCP message layer does not send a version number in each message. If required in the future, a new Capability and Settings Option (See Appendix B) could be defined to support version negotiation.

- 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, shown here as an 8-bit length.

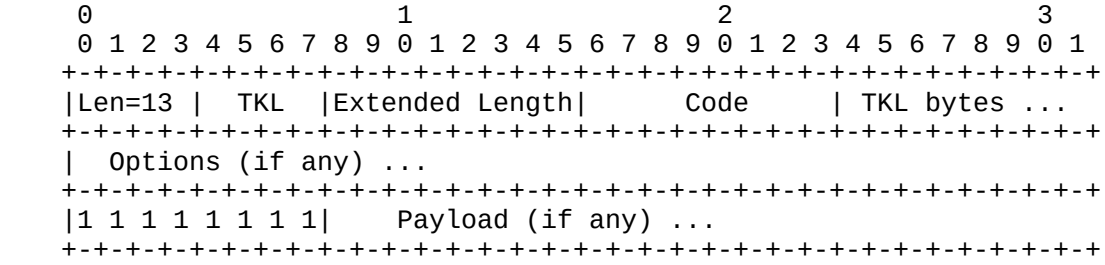


Figure 4: CoAP frame with 8-bit Extended Length field.

Length (Len): 4-bit unsigned integer. A value between 0 and 12 directly 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 on CoAP Options (see section 3.1 of [RFC7252]).

The following figures show the message format for the 0-bit, 16-bit, and the 32-bit variable length cases.

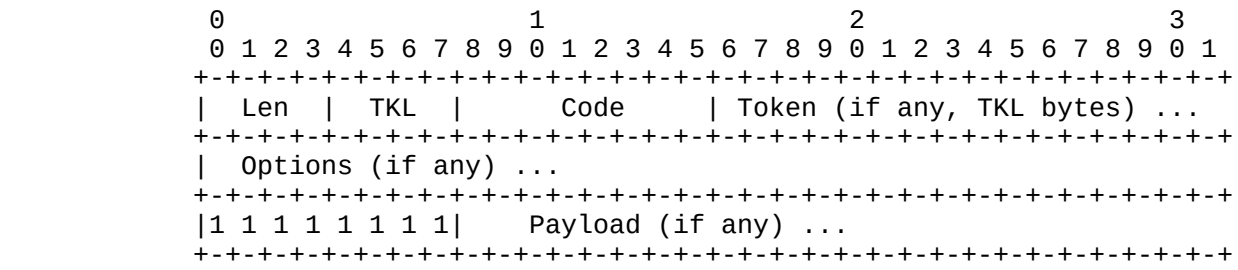


Figure 5: CoAP message format without an Extended Length field.

For example: A CoAP message just containing a 2.03 code with the token 7f and no options or payload would be encoded as shown in Figure 6.

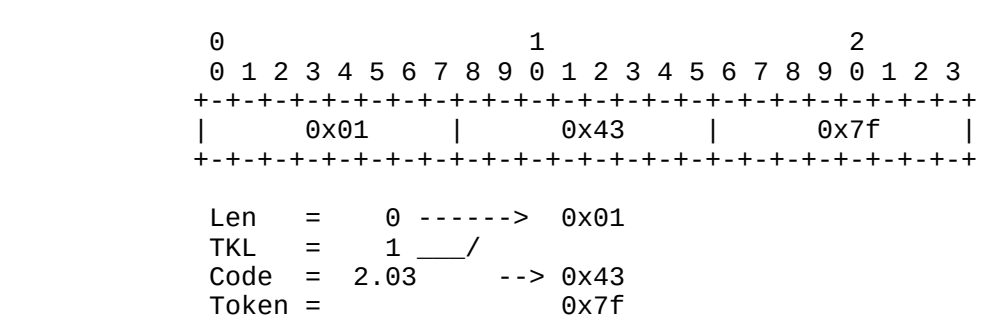


Figure 6: CoAP message with no options or payload.

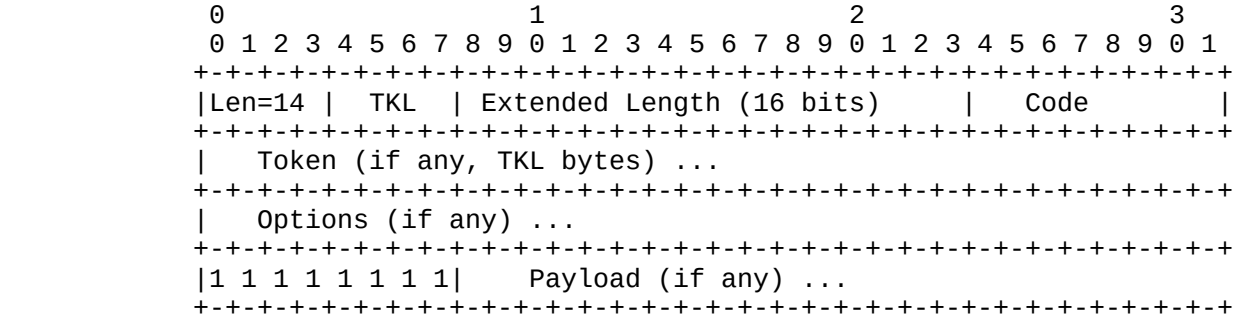


Figure 7: CoAP message format with 16-bit Extended Length field.

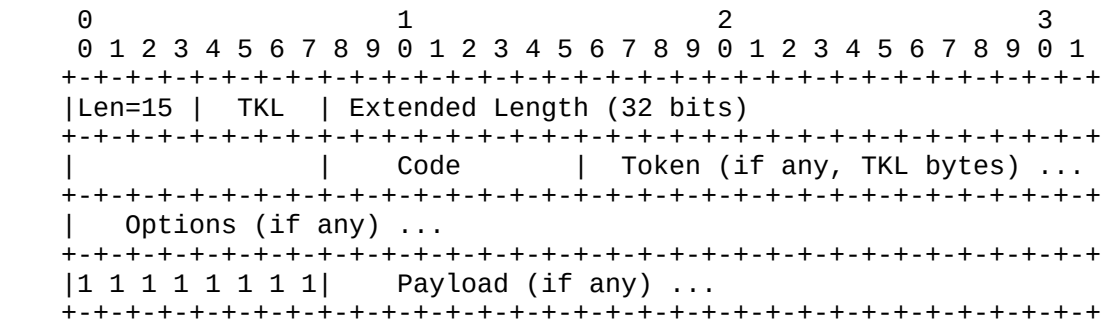


Figure 8: CoAP message format with 32-bit Extended Length field.

The semantics of the other CoAP header fields are left unchanged.

2.5. Message Transmission

CoAP requests and responses are exchanged asynchronously over the TCP/TLS 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.

Retransmission and deduplication of messages is provided by the TCP/TLS protocol.

3. CoAP over WebSockets

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 at a CoAP server that exposes a WebSocket endpoint (Figure 9). 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.

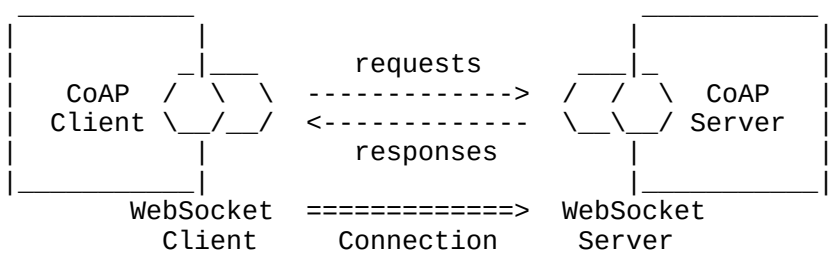


Figure 9: CoAP Client (WebSocket client) accesses CoAP Server (WebSocket server)

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. This means it is necessary for the client to identify both the WebSocket endpoint (identified by a "ws" or "wss" URI) and the path and query of the CoAP resource within that endpoint from the same URI. When the WebSocket protocol is used from a web page, the choices are more limited [RFC6454], but the challenge persists.

Section 6.2 defines a new "coap+ws" URI scheme that identifies both a WebSocket endpoint and a resource within that endpoint as follows:

```
          coap+ws://example.org/sensors/temperature?u=Cel
               \_____  _____/\__________  __________/
                     \/                 \/
                                    Uri-Path: "sensors"
ws://example.org/.well-known/coap   Uri-Path: "temperature"
                                    Uri-Query: "u=Cel"
```

Figure 10: The "coap+ws" URI Scheme

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 11), an SMS endpoint (a.k.a. mobile phone), or even another WebSocket endpoint. The client specifies the resource to be updated or retrieved in the Proxy-URI Option.

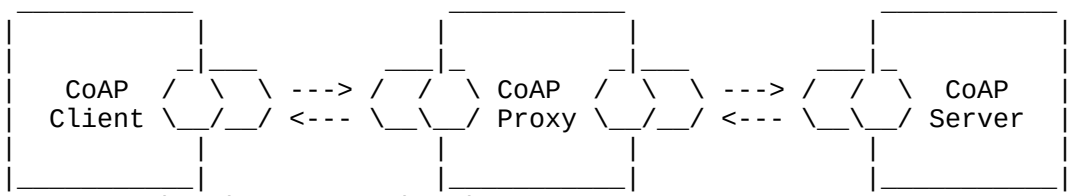


Figure 11: CoAP Client (WebSocket client) accesses CoAP Server (UDP server) via a CoAP proxy (WebSocket server/UDP client)

A third possible configuration is a CoAP server running inside a web browser (Figure 12). 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.

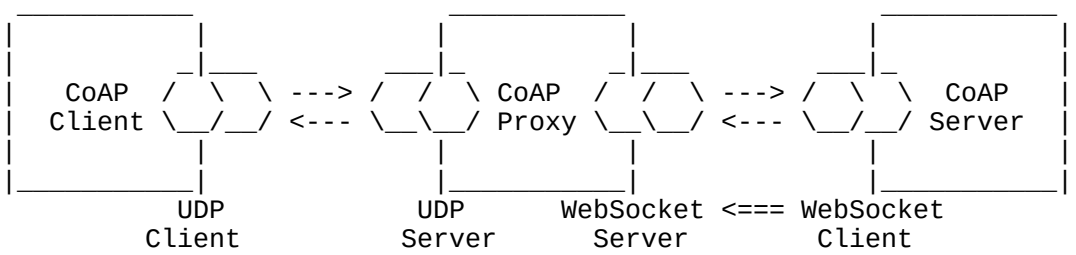


Figure 12: CoAP Client (UDP client) accesses sleepy CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

Further configurations are possible, including those where a WebSocket connection is established through an HTTP proxy.

CoAP over WebSockets is intentionally very similar to CoAP over UDP. Therefore, instead of presenting CoAP over WebSockets as a new protocol, this document specifies it as a series of deltas from [RFC7252].

3.1. Opening Handshake

Before CoAP requests and responses are exchanged, a WebSocket connection is established as defined in Section 4 of [RFC6455]. Figure 13 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. Any later, incompatible versions of CoAP or CoAP over WebSockets will use a different subprotocol name.

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 13: Example of an Opening Handshake

3.2. 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 14 is the same as the CoAP over TCP message format (see Section 2.4) with one restriction. The Length (Len) field MUST be set to zero because the WebSockets frame contains the length.

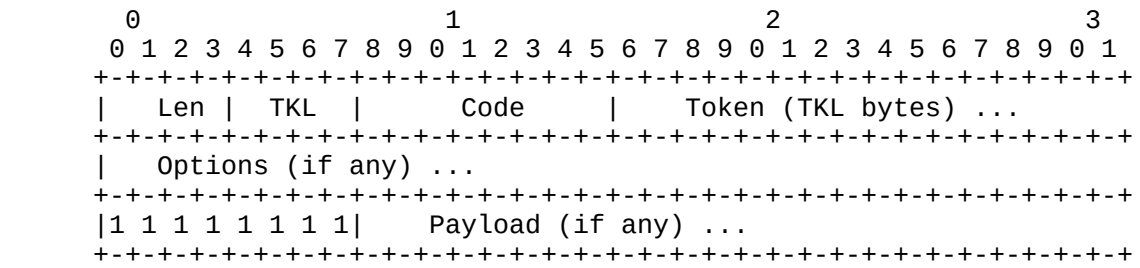


Figure 14: CoAP Message Format over WebSockets

The CoAP over TCP message format 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].

Empty messages (Code 0.00) MUST be ignored by the recipient (see also Section 4.4).

3.3. Message Transmission

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.

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.

3.4. Connection Health

When a 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), the connection between the WebSocket client and the WebSocket server may be lost or temporarily disrupted without the client being aware of it.

To check the health of the WebSocket connection (and thereby of all active requests, if any), a client can send a CoAP Ping Signaling message (Section 4.4). WebSocket Ping and unsolicited Pong frames as specified in Section 5.5 of [RFC6455] SHOULD NOT be used to ensure that redundant maintenance traffic is not transmitted.

There is no way to retransmit a request without creating a new one. Re-registering interest in a resource is permitted, but entirely unnecessary.

3.5. Closing the Connection

The WebSocket connection is closed as specified in Section 7 of [RFC6455].

All requests for which the CoAP client has not received a response yet are cancelled when the connection is closed.

4. Signaling

Signaling messages are introduced to allow peers to:

- Share characteristics such as maximum message size for the connection
- Shutdown the connection in an ordered fashion
- Terminate the 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, as adapted to the specific transport.)

4.1. Signaling Codes

A code in the 7.01-7.31 range indicates a Signaling message. Values in this range are assigned by the "CoAP Signaling Codes" sub-registry (see Section 8.1).

For each message, there is a sender and a peer receiving the message.

Payloads in Signaling messages are diagnostic payloads (see Section 5.5.2 of [RFC7252]), unless otherwise defined by a Signaling message option.

4.2. 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 8.2).

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 4.6). If the option is understood but cannot be processed, the option documents the behavior.

4.3. Capability and Settings Messages (CSM)

Capability and Settings messages (CSM) are used for two purposes:

- Each capability option advertises one capability of the sender to the recipient.

- Setting options indicate a setting that will be applied by the sender.

A Capability and Settings message MUST be sent by both endpoints at the start of the connection and MAY be sent at any other time by either endpoint over the lifetime of the connection.

Both capability and settings options are cumulative. A Capability and Settings message 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 Capability and Settings messages send a modified value.

These are not default values for the option as defined in Section 5.4.4 in [RFC7252]. A default value would mean that an empty Capability and Settings message would result in the option being set to its default value.

Capability and Settings messages are indicated by the 7.01 code (CSM).

4.3.1. Server-Name Setting Option

Number	Applies to	Name	Format	Length	Base Value
1	CSM	Server-Name	string	1-255	(see below)

A client can use the Server-Name critical option to indicate the default value for the Uri-Host Options in the messages that it sends to the server. It has the same restrictions as the Uri-Host Option (Section 5.10 of [RFC7252].

For TLS, the initial value for the Server-Name Option is given by the SNI value.

For Websockets, the initial value for the Server-Name Option is given by the HTTP Host header field.

4.3.2. Max-Message-Size Capability Option

The sender can use the Max-Message-Size elective option to indicate the maximum message size in bytes that it can receive.

Number	Applies to	Name	Format	Length	Base Value
2	CSM	Max-Message-Size	uint	0-4	1152

As per Section 4.6 of [RFC7252], the base value (and the value used when this option is not implemented) is 1152. A peer that relies on

this option being indicated with a certain minimum value will enjoy limited interoperability.

4.3.3. Block-wise Transfer Capability Option

Number	Applies to	Name	Format	Length	Base Value
4	CSM	Block-wise Transfer	empty	0	(none)

A sender can use the Block-wise Transfer elective 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 that supports block-wise transfers SHOULD 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 5).

4.4. Ping and Pong Messages

In CoAP over TCP, 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. In contrast, Ping and Pong messages are a bidirectional exchange.

Upon receipt of a Ping message, a single Pong message is returned with the identical token. 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).

4.4.1. Custody Option

Number	Applies to	Name	Format	Length	Base Value
2	Ping, Pong	Custody	empty	0	(none)

A peer replying to a Ping message can add a Custody elective option to the Pong message it returns. This option indicates that the application has processed all request/response messages that it has received in the present connection ahead of the Ping message that prompted the Pong message. (Note that there is no definition of specific application semantics of "processed", but there is an expectation that the sender of the Ping leading to the Pong with a Custody Option should be able to free buffers based on this indication.)

A Custody elective option can also be sent in a Ping message to explicitly request the return of a Custody Option in the Pong message. A peer is always free to indicate that it has finished processing all previous request/response messages by sending a Custody Option in a Pong message. A peer is also free NOT to send a Custody Option in case it is still processing previous request/response messages; however, it SHOULD delay its response to a Ping with a Custody Option until it can also return one.

4.5. Release Messages

A release message indicates that the sender does not want to continue maintaining the connection and opts for an orderly shutdown; the details are in the options. A diagnostic payload MAY be included. A release message will normally be replied to by the peer by closing the TCP/TLS connection. Messages may be in flight when the sender decides to send a Release message. The general expectation is that these will still be processed.

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:

Number	Applies to	Name	Format	Length	Base Value
2	Release	Bad-Server-Name	empty	0	(none)

The Bad-Server-Name elective option indicates that the default indicated by the CSM Server-Name Option is unlikely to be useful for this server.

Number	Applies to	Name	Format	Length	Base Value
4	Release	Alternate-Address	string	1-255	(none)

The Alternative-Address elective 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].

Number	Applies to	Name	Format	Length	Base Value
6	Release	Hold-Off	uint	0-3	(none)

The Hold-Off elective option indicates that the server is requesting that the peer not reconnect to it for the number of seconds given in the value.

4.6. Abort Messages

An abort message indicates that the sender is unable to continue maintaining the 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 SHOULD be included in the Abort message. Messages may be in flight 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:

Number	Applies to	Name	Format	Length	Base Value
2	Abort	Bad-CSM-Option	uint	0-2	(none)

The 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.

One reason for an sender to generate an abort message is a general syntax error in the byte stream received. No specific option has been defined for this, as the details of that syntax error are best left to a diagnostic payload.

4.7. Capability and Settings examples

An encoded example of a Ping message with a non-empty token is shown in Figure 15.

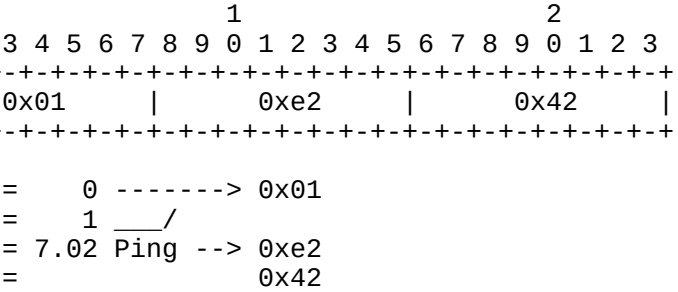


Figure 15: Ping Message Example

An encoded example of the corresponding Pong message is shown in Figure 16.

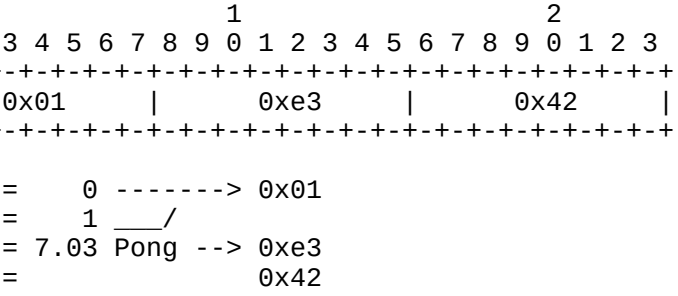


Figure 16: Pong Message Example

5. 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 byte stream transport. While this suggests that

the Block-wise transfer protocol [RFC7959] is also no longer needed, it remains applicable for a number of cases:

- large messages, such as firmware downloads, may cause undesired head-of-line blocking when a single TCP connection is used

- 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 allowed to contain a multiple of 1024 bytes (non-final BERT block) or more than 1024 bytes (final BERT block).

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 exponent (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.

5.1. Example: GET with BERT Blocks

Figure 17 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.

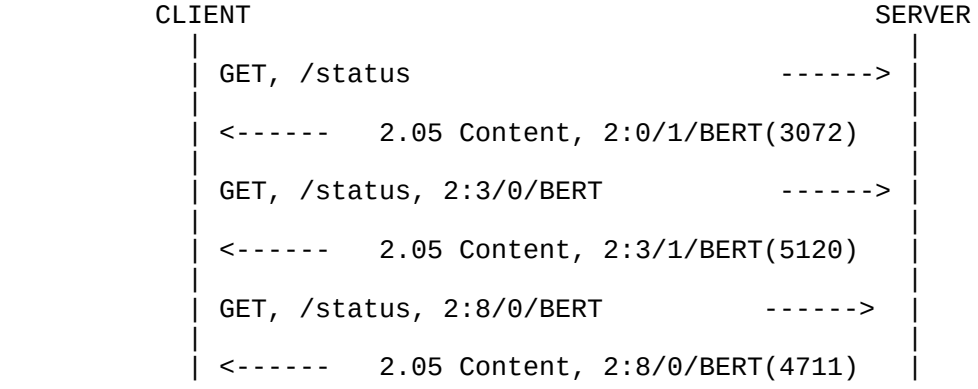


Figure 17: GET with BERT blocks.

5.2. Example: PUT with BERT Blocks

Figure 18 demonstrates a PUT exchange with BERT blocks.

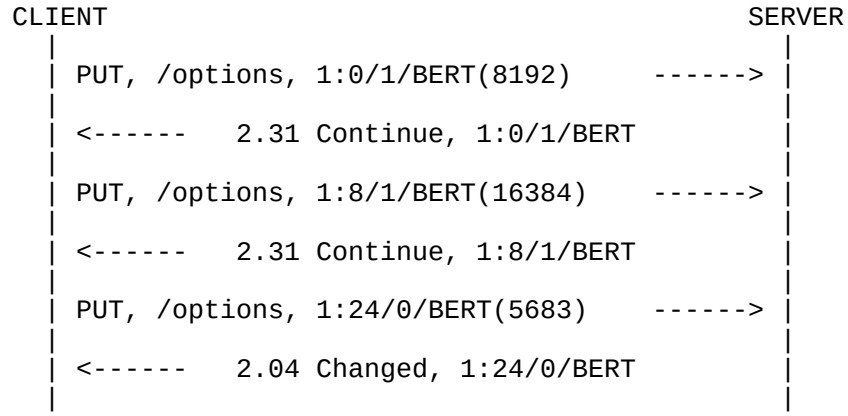


Figure 18: PUT with BERT blocks.

6. CoAP URIs

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

6.1. CoAP over TCP and TLS URIs

CoAP over TCP uses the "coap+tcp" URI scheme. CoAP over TLS uses the "coaps+tcp" scheme. The rules from Section 6 of [RFC7252] apply to both of these URI schemes.

[RFC7252], Section 8 (Multicast CoAP) is not applicable to these schemes.

Resources made available via one of the "coap+tcp" or "coaps+tcp" 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.

6.1.1. coap+tcp URI scheme

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

The semantics defined in [RFC7252], Section 6.1, apply to this URI scheme, with the following changes:

- The port subcomponent indicates the TCP port at which the CoAP server is located. (If it is empty or not given, then the default port 5683 is assumed, as with UDP.)

6.1.2. coaps+tcp URI scheme

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

The semantics defined in [RFC7252], Section 6.2, apply to this URI scheme, with the following changes:

- The port subcomponent indicates the TCP port at which the TLS server for the CoAP server is located. If it is empty or not given, then the default port 443 is assumed (this is different from the default port for "coaps", i.e., CoAP over DTLS over UDP).

- If a server does not support the Application-Layer Protocol Negotiation Extension (ALPN) [RFC7301] or wishes to accommodate 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 server MAY offer coaps+tcp endpoints on ports other than TCP port 5684, which MUST be ALPN enabled.

- For TCP ports other than port 5684, the client MUST use the ALPN extension to advertise the "coap" protocol identifier (see Section 8.7) in the list of protocols in its ClientHello. If the server selects and returns the "coap" protocol identifier using the ALPN extension in its ServerHello, then the connection succeeds. If the server either does not negotiate the ALPN extension or returns a no_application_protocol alert, the client MUST close the connection.

- For TCP port 5684, a client MAY use the ALPN extension to advertise the "coap" protocol identifier in the list of protocols in its ClientHello. If the server selects and returns the "coap" protocol identifier using the ALPN extension in its ServerHello, then the connection succeeds. If the server returns a no_application_protocol alert, then the client MUST close the connection. If the server does not negotiate the ALPN extension, then coaps+tcp is implicitly selected.

- For TCP port 5684, if the client does not use the ALPN extension to negotiate the protocol, then coaps+tcp is implicitly selected.

6.2. CoAP over WebSockets URIs

For the first configuration discussed in Section 3, this document defines two new URIs schemes that can be used for identifying CoAP resources and providing a means of locating these resources: "coap+ws" and "coap+wss".

Similar to the "coap" and "coaps" schemes, the "coap+ws" and "coap+wss" schemes organize resources hierarchically under a CoAP origin server. The key difference is that the server is potentially reachable on a WebSocket endpoint instead of a UDP endpoint.

The WebSocket endpoint is identified by a "ws" or "wss" URI that is composed of the authority part of the "coap+ws" or "coap+wss" URI, respectively, and the well-known path "/.well-known/coap" [RFC5785]. The path and query parts of a "coap+ws" or "coap+wss" URI identify a resource within the specified endpoint which can be operated on by the methods defined by CoAP.

The syntax of the "coap+ws" and "coap+wss" URI schemes is specified below in Augmented Backus-Naur Form (ABNF) [RFC5234]. The definitions of "host", "port", "path-abempty" and "query" are the same as in [RFC3986].

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

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

The port component is OPTIONAL; the default for "coap+ws" is port 80, while the default for "coap+wss" is port 443.

Fragment identifiers are not part of the request URI and thus MUST NOT be transmitted in a WebSocket handshake or in the URI options of a CoAP request.

6.2.1. Decomposing and Composing URIs

The steps for decomposing a "coap+ws" or "coap+wss" URI into CoAP options are the same as specified in Section 6.4 of [RFC7252] with the following changes:

- The <scheme> component MUST be "coap+ws" or "coap+wss" when converted to ASCII lowercase.

- A Uri-Host Option MUST only be included in a request when the <host> component does not equal the uri-host component in the Host header field in the WebSocket handshake.

- A Uri-Port Option MUST only be included in a request if |port| does not equal the port component in the Host header field in the WebSocket handshake.

The steps to construct a URI from a request's options are changed accordingly.

7. Security Considerations

The security considerations of [RFC7252] apply.

TLS version 1.2 or higher is mandatory-to-implement and MUST be enabled by default. An endpoint MAY immediately abort a CoAP over TLS connection that does not meet this requirement (see Section 4.6) and SHOULD include a diagnostic payload.

The TLS usage guidance in [RFC7925] SHOULD be followed.

TLS does not protect the TCP header. This may, for example, allow an on-path adversary to terminate a TCP connection prematurely by spoofing a TCP reset message.

CoAP over WebSockets and CoAP over TLS-secured WebSockets do not introduce additional security issues beyond CoAP and DTLS-secured CoAP respectively [RFC7252]. The security considerations of [RFC6455] apply.

7.1. 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.

- SNI vs. Server-Name: Any security negotiated in the TLS handshake is for the SNI name exchanged in the TLS handshake and checked against the certificate provided by the server. The Server-Name Option cannot be used to extend these security properties to the additional server name.

8. IANA Considerations

8.1. 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 7.01-7.31, 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 [RFC5226].

8.2. CoAP Signaling Option Numbers Registry

IANA is requested to create a sub-registry for signaling options similar to the CoAP Option Numbers Registry (Section 12.2 of [RFC7252]), with the single change that a fourth column is added to the sub-registry that is one of the codes in the Signaling Codes subregistry (Section 8.1).

The name of this sub-registry is "CoAP Signaling Option Numbers".

Initial entries in this sub-registry are as follows:

Number	Applies to	Name	Reference
1	CSM	Server-Name	[RFCthis]
2	CSM	Max-Message-Size	[RFCthis]
4	CSM	Block-wise-Transfer	[RFCthis]
2	Ping, Pong	Custody	[RFCthis]
2	Release	Bad-Server-Name	[RFCthis]
4	Release	Alternative-Address	[RFCthis]
6	Release	Hold-Off	[RFCthis]
2	Abort	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].

8.3. 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

8.4. 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

8.5. URI Scheme Registration

This document registers two new URI schemes, namely "coap+tcp" and "coaps+tcp", for the use of CoAP over TCP and for CoAP over TLS over TCP, respectively. The "coap+tcp" and "coaps+tcp" URI schemes can thus be compared to the "http" and "https" URI schemes.

The syntax of the "coap" and "coaps" URI schemes is specified in Section 6 of [RFC7252] and the present document re-uses their semantics for "coap+tcp" and "coaps+tcp", respectively, with the exception that TCP, or TLS over TCP is used as a transport protocol.

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

8.5.1. coap+ws

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+ws". The registration request complies with [RFC4395].

URL scheme name.
coap+ws

Status.
Permanent

URI scheme syntax.
Defined in Section N of [RFCthis]

URI scheme semantics.
The "coap+ws" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket protocol.

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol.

Interoperability considerations.
None.

Security Considerations.
See Section N of [RFCthis]

Contact.
IETF chair <chair@ietf.org>

Author/Change controller.
IESG <iesg@ietf.org>

References.
[RFCthis]

8.5.2. coap+wss

This document requests the registration of the Uniform Resource Identifier (URI) scheme "coap+wss". The registration request complies with [RFC4395].

URL scheme name.
coap+wss

Status.
Permanent

URI scheme syntax.
Defined in Section N of [RFCthis]

URI scheme semantics.
The "coap+ws" URI scheme provides a way to identify resources that are potentially accessible over the Constrained Application Protocol (CoAP) using the WebSocket protocol secured with Transport Layer Security (TLS).

Encoding considerations.
The scheme encoding conforms to the encoding rules established for URIs in [RFC3986], i.e., internationalized and reserved characters are expressed using UTF-8-based percent-encoding.

Applications/protocols that use this URI scheme name.
The scheme is used by CoAP endpoints to access CoAP resources using the WebSocket protocol secured with TLS.

Interoperability considerations.
None.

Security Considerations.
See Section N of [RFCthis]

Contact.
IETF chair <chair@ietf.org>

Author/Change controller.
IESG <iesg@ietf.org>

References.
[RFCthis]

8.6. 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.

8.7. 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]

8.8. 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]

9. References

9.1. Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, DOI 10.17487/RFC0793, September 1981, <http://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, <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>.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", RFC 4395, DOI 10.17487/RFC4395, February 2006, <http://www.rfc-editor.org/info/rfc4395>.

[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>.

[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>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <http://www.rfc-editor.org/info/rfc5785>.

[RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, DOI 10.17487/RFC6455, December 2011, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc7301>.

[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>.

[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>.

[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, <http://www.rfc-editor.org/info/rfc7925>.

9.2. Informative References

[HomeGateway] Eggert, L., "An experimental study of home gateway characteristics", Proceedings of the 10th annual conference on Internet measurement, 2010.

[LWM2M] Open Mobile Alliance, "Lightweight Machine to Machine Technical Specification Candidate Version 1.0", April 2016, <http://technical.openmobilealliance.org/Technical/Release_Program/docs/LightweightM2M/V1_0-20160407-C/OMA-TS-LightweightM2M-V1_0-20160407-C.pdf>.

[RFC0768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, DOI 10.17487/RFC0768, August 1980, <http://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, <http://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, <http://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, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, DOI 10.17487/RFC6454, December 2011, <http://www.rfc-editor.org/info/rfc6454>.

[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>.

[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>.

Appendix A. Updates to RFC7641 Observing Resources in the Constrained Application Protocol (CoAP)

A.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.

A.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 4.6) and SHOULD include a diagnostic payload.

A.3. 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 connection, 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.

Appendix B. Negotiating Protocol Versions

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.

In contrast to the message layer for UDP and DTLS, the CoAP over TCP message format does not include a version number. If version negotiation needs to be addressed in the future, then Capability and Settings have been specifically designed to enable such a potential feature.

Appendix C. CoAP over WebSocket Examples

This section gives examples for the first two configurations discussed in Section 3.

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 19 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. 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.

4. It waits for the server to return a response.

5. The CoAP client uses the connection for further requests, or the connection is closed.

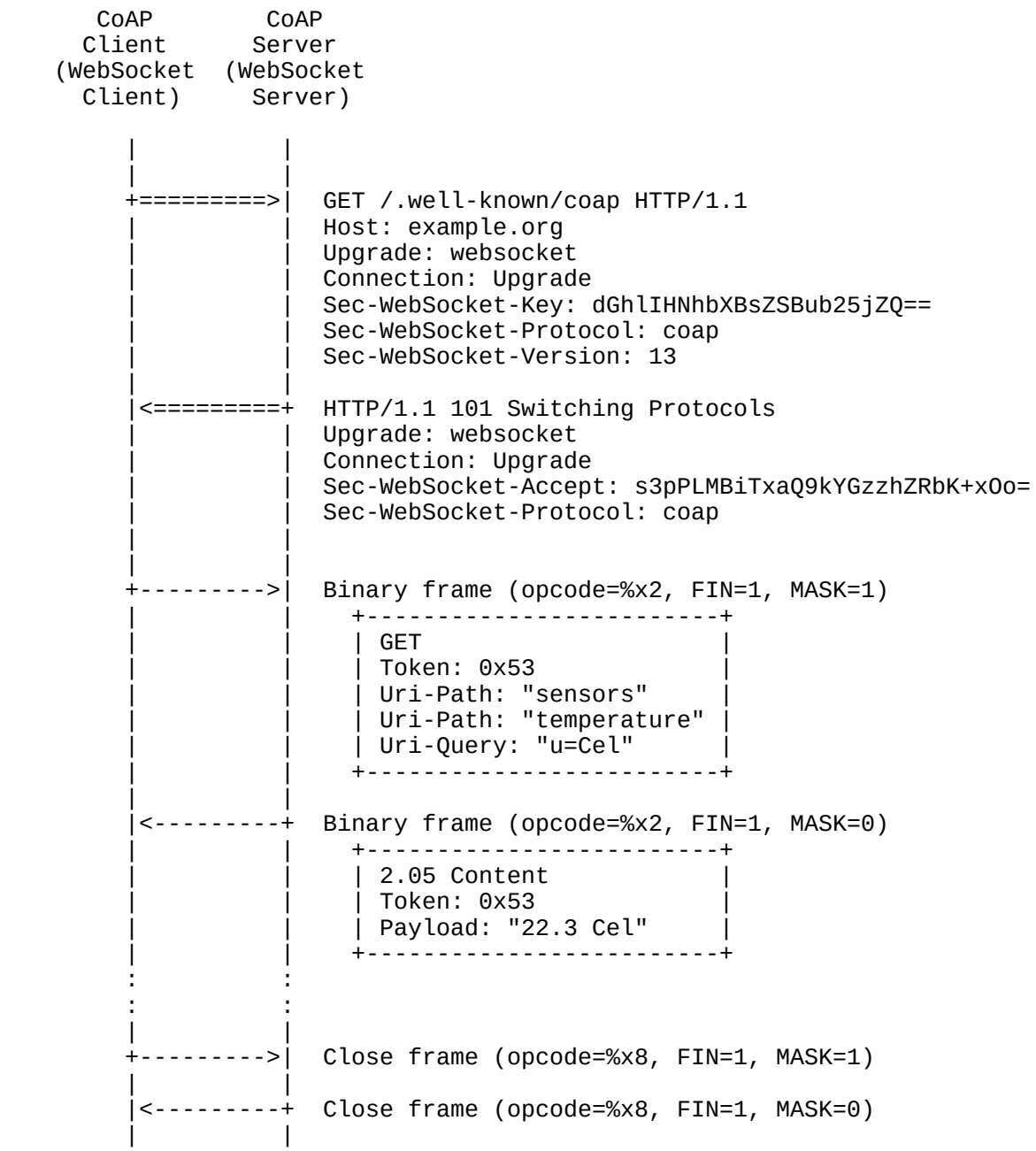


Figure 19: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

Figure 20 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.

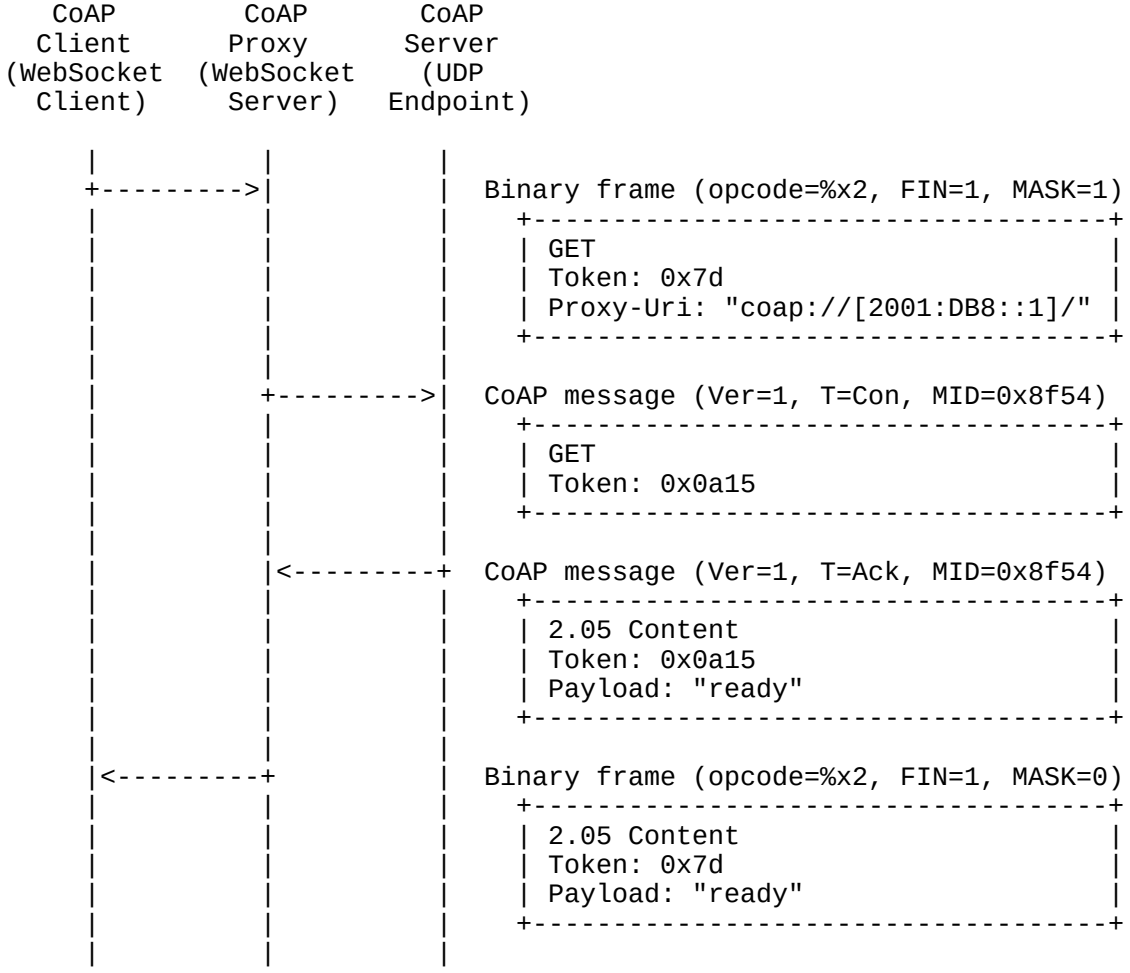


Figure 20: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSockets-enabled CoAP proxy

Appendix D. Change Log

The RFC Editor is requested to remove this section at publication.

D.1. Since draft-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

D.2. Since draft-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

D.3. Since draft-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 client to send messages before receiving server CSM

Acknowledgements

We would like to thank Stephen Berard, Geoffrey Cristallo, Olivier Delaby, Christian Groves, Nadir Javed, Michael Koster, Matthias Kovatsch, Achim Kraus, David Navarro, Szymon Sasin, Zach Shelby, Andrew Summers, Julien Vermillard, and Gengyu Wei for their feedback.

Contributors

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

Teemu Savolainen
Nokia Technologies
Hatanpaan valtatie 30
Tampere FI-33100
Finland

Email: teemu.savolainen@nokia.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)
Microsoft
One Microsoft Way
Redmond 98052
United States of America

Email: brian.raymor@microsoft.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.

Table of Contents

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:

- 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.

- 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:

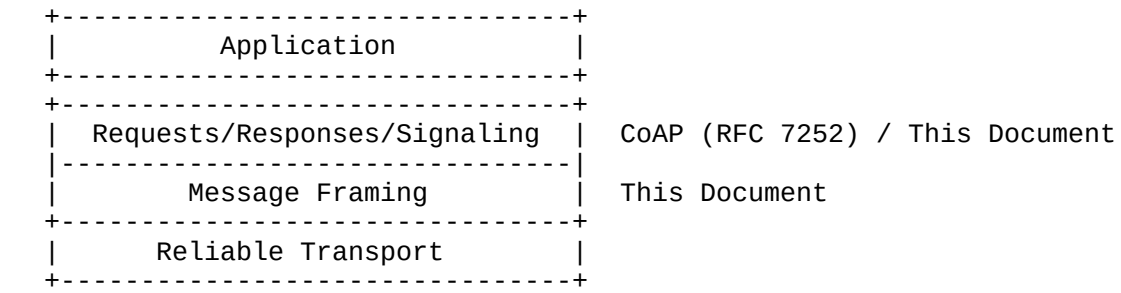


Figure 1: Layering of CoAP over Reliable Transports

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.

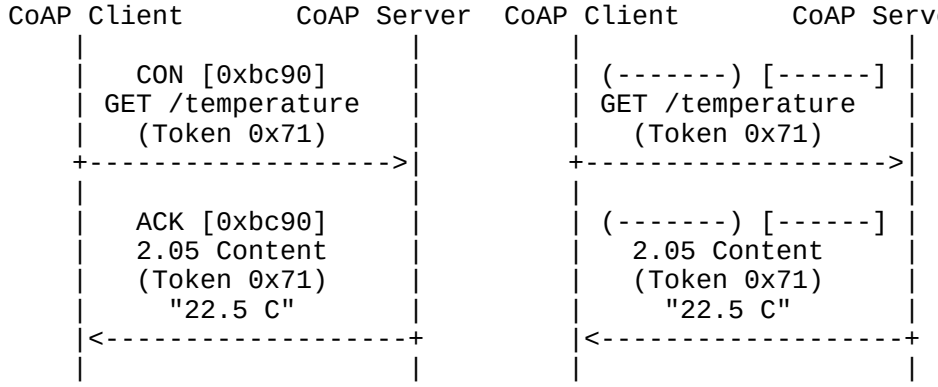


Figure 2: Comparison between CoAP over unreliable and reliable transport

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.

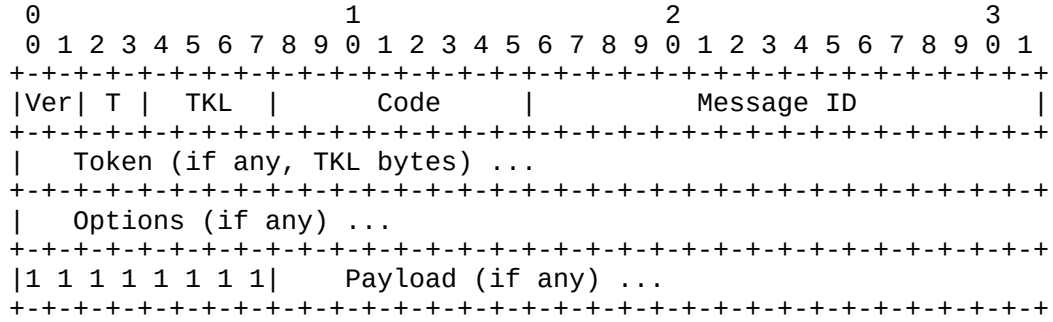


Figure 3: RFC 7252 defined CoAP Message Format

The CoAP over TCP message format is very similar to the format specified for CoAP over UDP. The differences are as follows:

- 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.

- 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.

- 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.

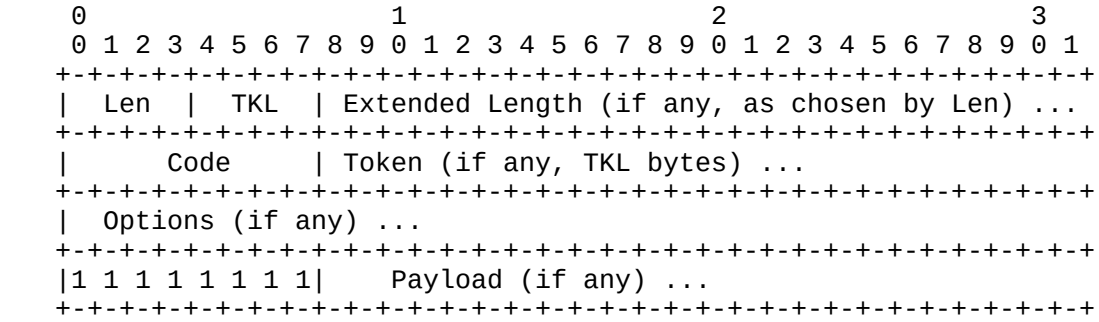


Figure 4: CoAP frame for reliable transports

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.

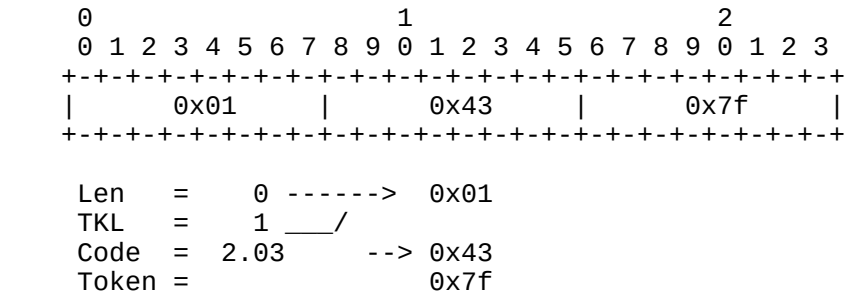


Figure 5: CoAP message with no options or payload

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.

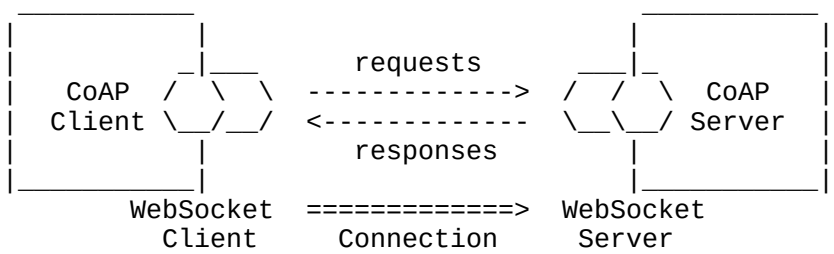


Figure 6: CoAP Client (WebSocket client) accesses CoAP Server (WebSocket server)

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.

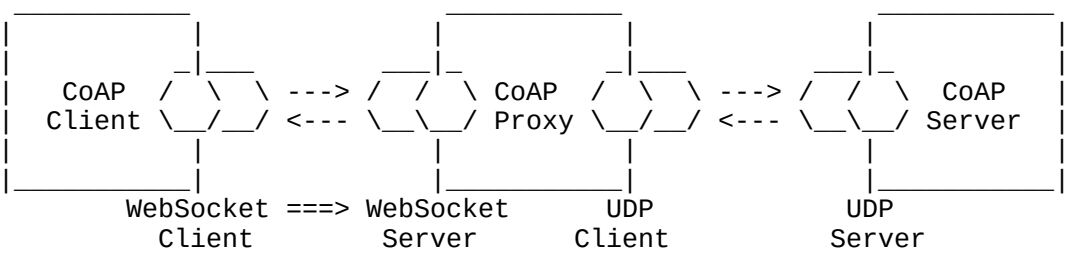


Figure 7: CoAP Client (WebSocket client) accesses CoAP Server (UDP server) via a CoAP proxy (WebSocket server/UDP client)

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.

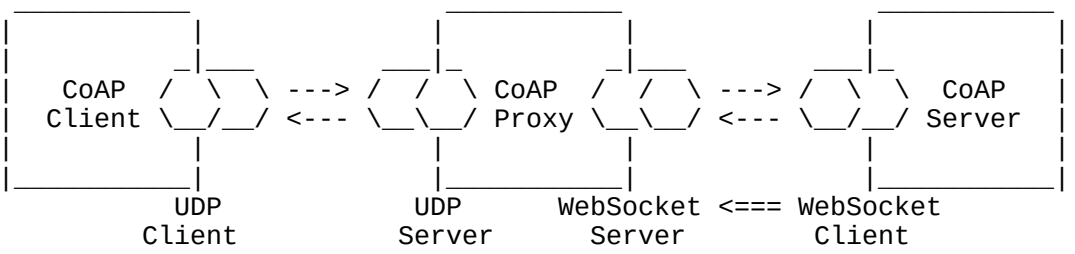


Figure 8: CoAP Client (UDP client) accesses CoAP Server (WebSocket client) via a CoAP proxy (UDP server/WebSocket server)

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.

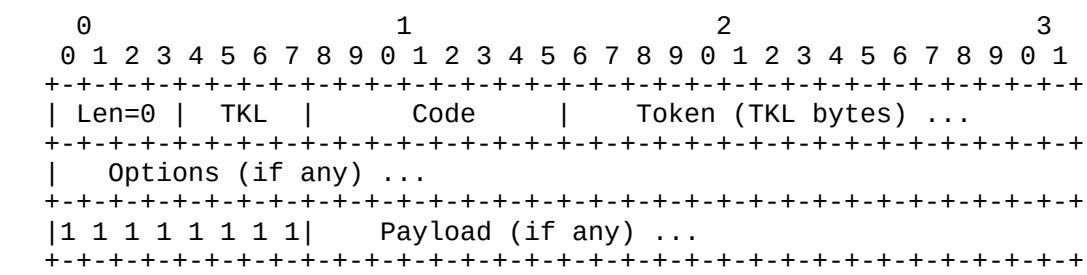


Figure 10: CoAP Message Format over WebSockets

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:

- Learn related characteristics, such as maximum message size for the connection
- Shut down the connection in an orderly fashion

- 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:

- Each capability option indicates one capability of the sender to the recipient.

- 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.

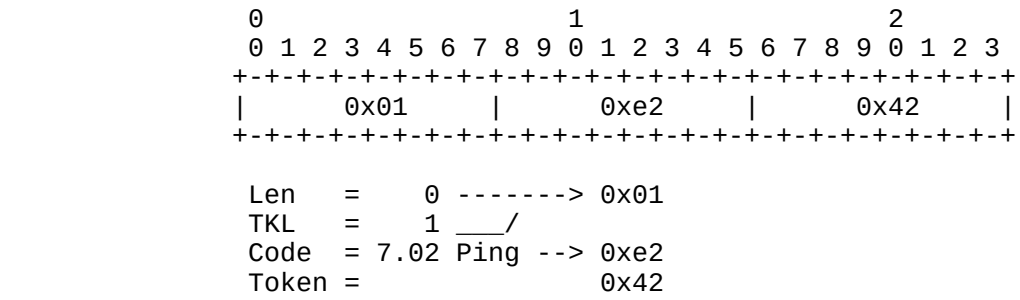


Figure 11: Ping Message Example

An encoded example of the corresponding Pong message is shown in Figure 12.

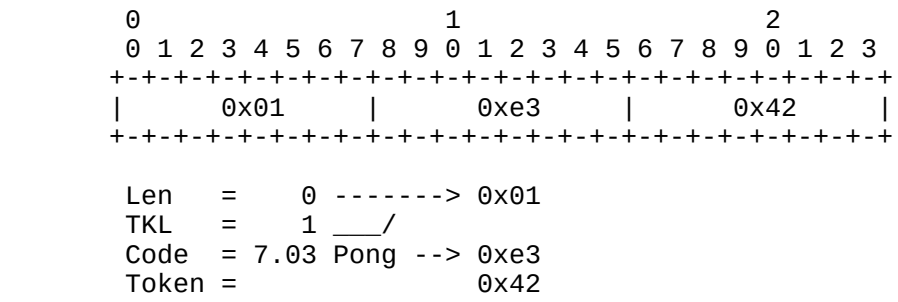


Figure 12: Pong Message Example

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:

- large messages, such as firmware downloads, may cause undesired head-of-line blocking when a single transport connection is used
- 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.

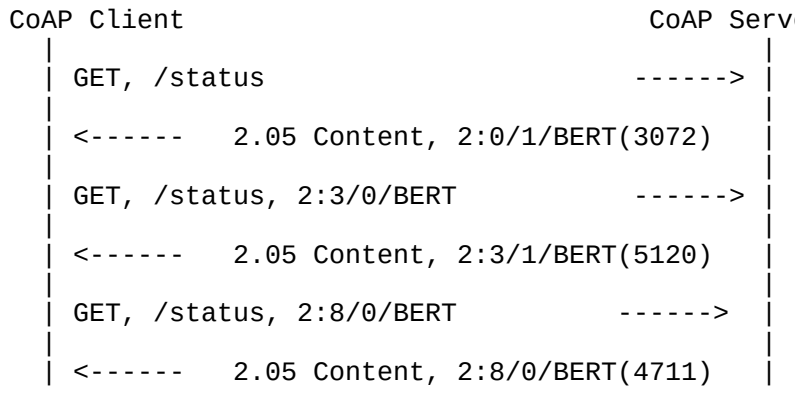


Figure 13: GET with BERT blocks

6.2. Example: PUT with BERT Blocks

Figure 14 demonstrates a PUT exchange with BERT blocks.

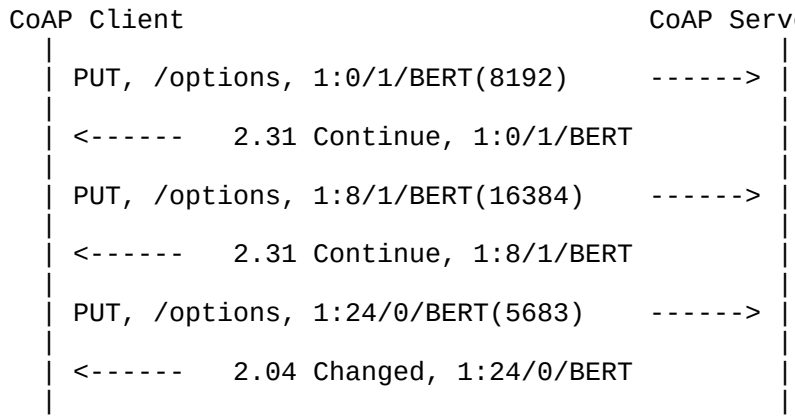


Figure 14: PUT with BERT blocks

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:

- the "coap+tcp" URI scheme for CoAP over TCP
- the "coaps+tcp" URI scheme for CoAP over TCP secured by TLS
- the "coap+ws" URI scheme for CoAP over WebSockets
- 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:

- 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:

- 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.

- 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.

- 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.

- 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.

- 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:

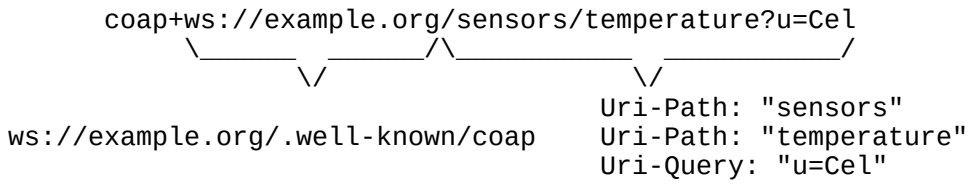


Figure 15: The "coap+ws" URI Scheme

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.

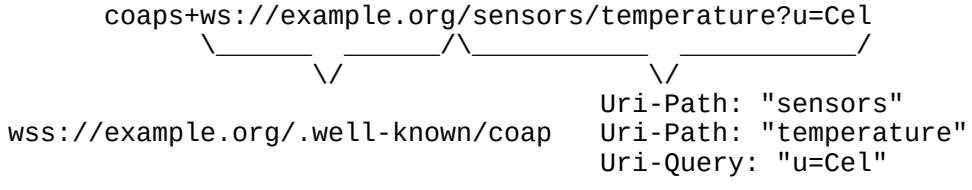


Figure 16: The "coaps+ws" URI Scheme

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 [RFC7252].

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:

- Whether the option is critical or elective, as determined by the Option Number.
- Whether the option is repeatable.
- The format and length of the option’s value.
- 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.

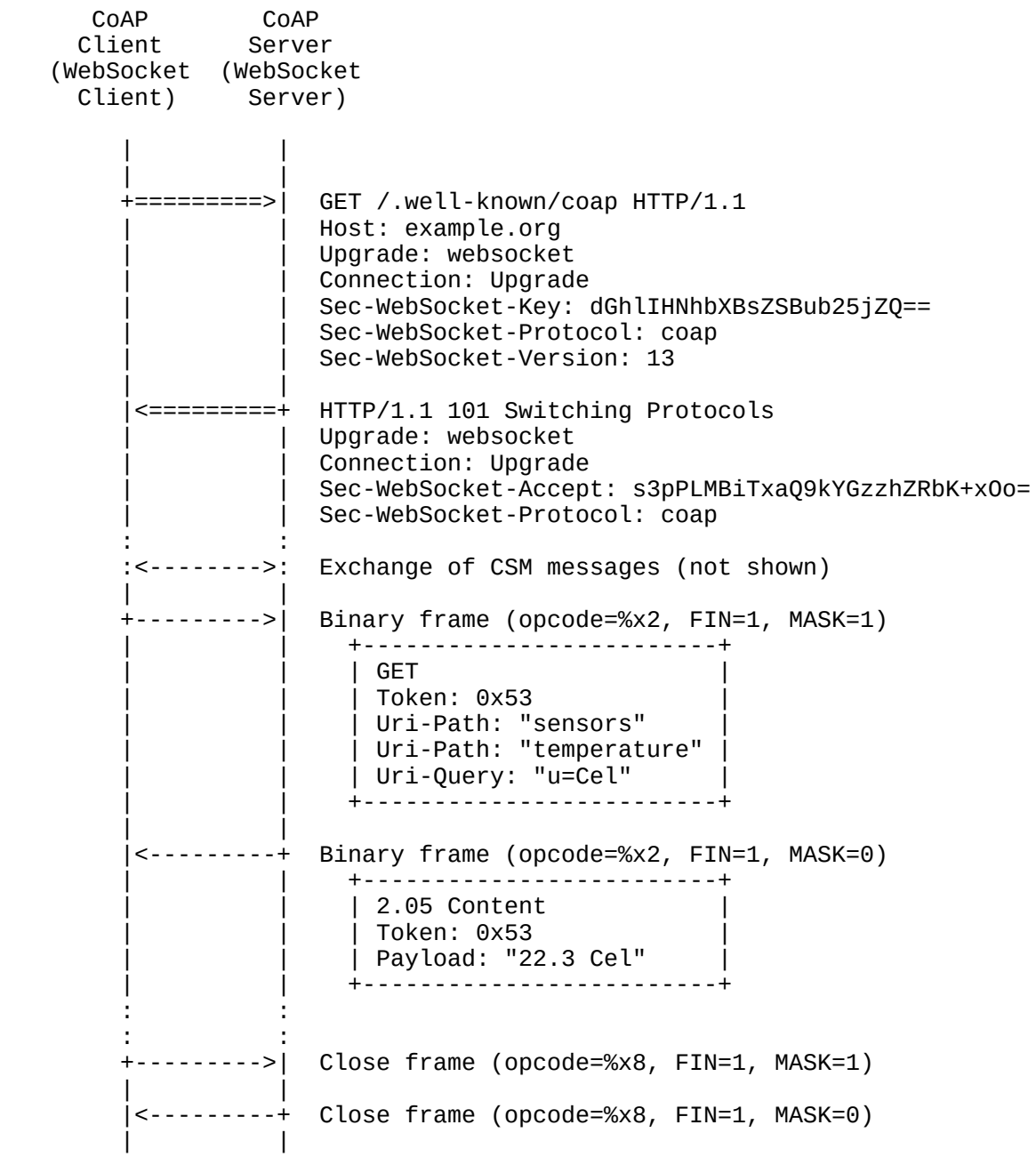


Figure 17: A CoAP client retrieves the representation of a resource identified by a "coap+ws" URI

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.

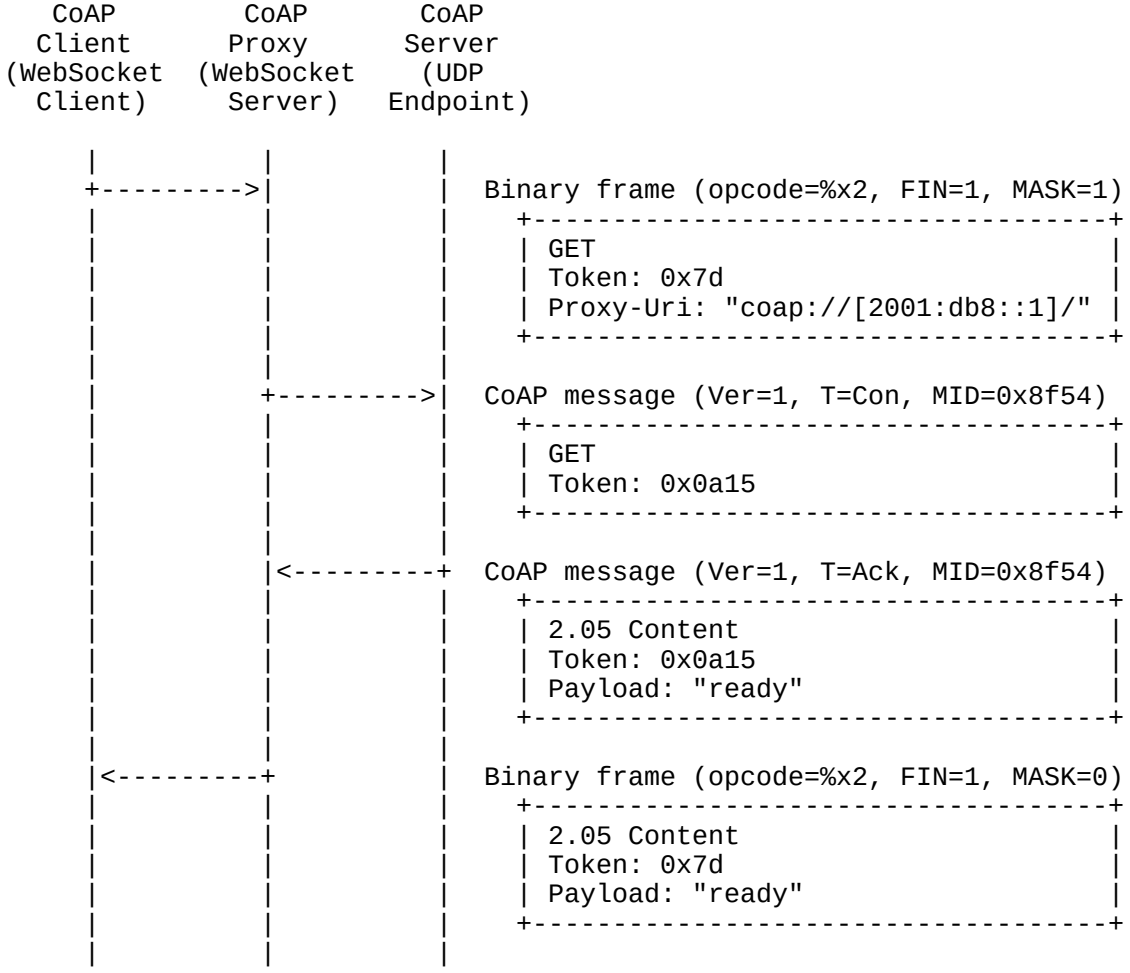


Figure 18: A CoAP client retrieves the representation of a resource identified by a "coap" URI via a WebSocket-enabled CoAP proxy

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: January 23, 2020

Z. Shelby
ARM
M. Koster
SmartThings
C. Groves

J. Zhu
Huawei
B. Silverajan, Ed.
Tampere University
July 22, 2019

Dynamic Resource Linking for Constrained RESTful Environments
draft-ietf-core-dynlink-10

Abstract

This specification defines Link Bindings, which provide dynamic linking of state updates between resources, either on an endpoint or between endpoints, for systems using CoAP (RFC7252). This specification also defines Conditional Notification Attributes that work with Link Bindings or with CoAP Observe (RFC7641).

Editor note

The git repository for the draft is found at https://github.com/core-wg/dynlink

Status of This Memo

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

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

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

This Internet-Draft will expire on January 23, 2020.

Table of Contents

1. Introduction

IETF Standards for machine to machine communication in constrained environments describe a REST protocol [RFC7252] and a set of related information standards that may be used to represent machine data and machine metadata in REST interfaces. CoRE Link-format [RFC6690] is a standard for doing Web Linking [RFC8288] in constrained environments.

This specification introduces the concept of a Link Binding, which defines a new link relation type to create a dynamic link between resources over which state updates are conveyed. Specifically, a Link Binding is a unidirectional link for binding the states of source and destination resources together such that updates to one are sent over the link to the other. CoRE Link Format representations are used to configure, inspect, and maintain Link Bindings. This specification additionally defines Conditional Notification Attributes for use with Link Bindings and with CoRE Observe [RFC7641].

2. Terminology

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

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC8288] and [RFC6690]. This specification makes use of the following additional terminology:

Link Binding: A unidirectional logical link between a source resource and a destination resource, over which state information is synchronized.

State Synchronization: Depending on the binding method (Polling, Observe, Push) different REST methods may be used to synchronize the resource values between a source and a destination. The process of using a REST method to achieve this is defined as "State Synchronization". The endpoint triggering the state synchronization is the synchronization initiator.

Notification Band: A resource value range that results in state sychronization. The value range may be bounded by a minimum and maximum value or may be unbounded having either a minimum or maximum value.

3. Conditional Notification Attributes

3.1. Attribute Definitions

This specification defines Conditional Notification Attributes, which provide for fine-grained control of notification and state synchronization when using CoRE Observe [RFC7641] or Link Bindings (see Section 4). Conditional Notification Attributes define the conditions that trigger a notification.

When resource interfaces following this specification are made available over CoAP, the CoAP Observation mechanism [RFC7641] MAY also be used to observe any changes in a resource, and receive asynchronous notifications as a result. A resource marked as Observable in its link description SHOULD support these Conditional Notification Attributes.

The set of parameters defined here allow a client to control how often a client is interested in receiving notifications and how much a resource value should change for the new representation to be interesting.

One or more Notification Attributes MAY be included as query parameters in an Observe request.

These attributes are defined below:

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

Table 1: Conditional Notification Attributes

Conditional Notification Attributes SHOULD be evaluated on all potential notifications from a resource, whether resulting from an internal server-driven sampling process or from external update requests to the server.

Note: In this draft, we assume that there are finite quantization effects in the internal or external updates to the value of a resource; specifically, that a resource may be updated at any time with any valid value. We therefore avoid any continuous-time assumptions in the description of the Conditional Notification Attributes and instead use the phrase "sampled value" to refer to a member of a sequence of values that may be internally observed from the resource state over time.

3.1.1. Minimum Period (pmin)

When present, the minimum period indicates the minimum time, in seconds, between two consecutive notifications (whether or not the resource value has changed). In the absence of this parameter, the minimum period is up to the server. The minimum period MUST be greater than zero otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

A server MAY report the last sampled value that occured during the pmin interval, after the pmin interval expires.

Note: Due to finite quantization effects, the time between notifications may be greater than pmin even when the sampled value changes within the pmin interval. Pmin may or may not be used to drive the internal sampling process.

3.1.2. Maximum Period (pmax)

When present, the maximum period indicates the maximum time, in seconds, between two consecutive notifications (whether or not the resource value has changed). In the absence of this parameter, the maximum period is up to the server. The maximum period MUST be greater than zero and MUST be greater than the minimum period parameter (if present) otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

3.1.3. Change Step (st)

When present, the change step indicates how much the value of a resource SHOULD change before triggering a notification, compared to the value of the previous notification. Upon reception of a query including the st attribute, the most recently sampled value of the resource is reported, and then set as the last reported value (last_rep_v). When a subsequent sample or update of the resource value differs from the last reported value by an amount, positive or negative, greater than or equal to st, and the time for pmin has elapsed since the last notification, a notification is sent and the last reported value is updated to the value sent in the notification. The change step MUST be greater than zero otherwise the receiver MUST return a CoAP error code 4.00 "Bad Request" (or equivalent).

The Change Step parameter can only be supported on resources with a scalar numeric value.

Note: Due to sampling and other constraints, e.g. pmin, the resource value received in two sequential notifications may differ by more than st.

3.1.4. Greater Than (gt)

When present, Greater Than indicates the upper limit value the sampled value SHOULD cross before triggering a notification. A notification is sent whenever the sampled value crosses the specified upper limit value, relative to the last reported value, and the time fpr pmin has elapsed since the last notification. The sampled value is sent in the notification. If the value continues to rise, no notifications are generated as a result of gt. If the value drops below the upper limit value then a notification is sent, subject again to the pmin time.

The Greater Than parameter can only be supported on resources with a scalar numeric value.

3.1.5. Less Than (lt)

When present, Less Than indicates the lower limit value the resource value SHOULD cross before triggering a notification. A notification is sent when the samples value crosses the specified lower limit value, relative to the last reported value, and the time fpr pmin has elapsed since the last notification. The sampled value is sent in the notification. If the value continues to fall no notifications are generated as a result of lt. If the value rises above the lower limit value then a new notification is sent, subject to the pmin time..

The Less Than parameter can only be supported on resources with a scalar numeric value.

3.1.6. Notification Band (band)

The notification band attribute allows a bounded or unbounded (based on a minimum or maximum) value range that may trigger multiple notifications. This enables use cases where different ranges results in differing behaviour. For example: monitoring the temperature of machinery. Whilst the temperature is in the normal operating range only periodic observations are needed. However as the temperature moves to more abnormal ranges more frequent synchronization/reporting may be needed.

Without a notification band, a transition across a less than (lt), or greater than (gt) limit only generates one notification. This means that it is not possible to describe a case where multiple notifications are sent so long as the limit is exceeded.

The band attribute works as a modifier to the behaviour of gt and lt. Therefore, if band is present in a query, gt, lt or both, MUST be included.

When band is present with the lt attribute, it defines the lower bound for the notification band (notification band minimum). Notifications occur when the resource value is equal to or above the notification band minimum. If lt is not present there is no minimum value for the band.

When band is present with the gt attribute, it defines the upper bound for the notification band (notification band maximum). Notifications occur when the resource value is equal to or below the notification band maximum. If gt is not present there is no maximum value for the band.

If band is present with both the gt and lt attributes, notification occurs when the resource value is greater than or equal to gt or when the resource value is less than or equal to lt.

If a band is specified in which the value of gt is less than that of lt, in-band notification occurs. That is, notification occurs whenever the resource value is between the gt and lt values, including equal to gt or lt.

If the band is specified in which the value of gt is greater than that of lt, out-of-band notification occurs. That is, notification occurs when the resource value not between the gt and lt values, excluding equal to gt and lt.

The Notification Band parameter can only be supported on resources with a scalar numeric value.

3.2. Server processing of Conditional Notification Attributes

Pmin, pmax, st, gt, lt and band may be present in the same query. Howver, they are not defined at multiple prioritization levels. The server sends a notification whenever any of the parameter conditions are met, upon which it updates its last notification value and time to prepare for the next notification. Only one notification occurs when there are multiple conditions being met at the same time. The reference code below illustrates the logic to determine when a notification is to be sent.

```
bool notifiable( Resource * r ) {

#define BAND r->band
#define SCALAR_TYPE ( num_type == r->type )
#define STRING_TYPE ( str_type == r->type )
#define BOOLEAN_TYPE ( bool_type == r->type )
#define PMIN_EX ( r->last_sample_time - r->last_rep_time >= r->pmin )
#define PMAX_EX ( r->last_sample_time - r->last_rep_time > r->pmax )
#define LT_EX ( r->v < r->lt ^ r->last_rep_v < r->lt )
#define GT_EX ( r->v > r->gt ^ r->last_rep_v > r->gt )
#define ST_EX ( abs( r->v - r->last_rep_v ) >= r->st )
#define IN_BAND ( ( r->gt <= r->v && r->v <= r->lt ) || ( r->lt <= r->gt && r->g
t <= r->v ) || ( r->v <= r->lt && r->lt <= r->gt ) )
#define VB_CHANGE ( r->vb != r->last_rep_vb )
#define VS_CHANGE ( r->vs != r->last_rep_vs )

  return (
    PMIN_EX &&
    ( SCALAR_TYPE ?
      ( ( !BAND && ( GT_EX || LT_EX || ST_EX || PMAX_EX ) ) ||
        ( BAND && IN_BAND && ( ST_EX || PMAX_EX) ) )
    : STRING_TYPE ?
      ( VS_CHANGE || PMAX_EX )
    : BOOLEAN_TYPE ?
      ( VB_CHANGE || PMAX_EX )
    : false )
  );
}
```

Figure 1: Code logic for conditional notification attribute interactions

4. Link Bindings

In a M2M RESTful environment, endpoints may directly exchange the content of their resources to operate the distributed system. For example, a light switch may supply on-off control information that may be sent directly to a light resource for on-off control. Beforehand, a configuration phase is necessary to determine how the resources of the different endpoints are related to each other. This can be done either automatically using discovery mechanisms or by means of human intervention and a so-called commissioning tool.

In this specification such an abstract relationship between two resources is defined, called a Link Binding. The configuration phase necessitates the exchange of binding information, so a format recognized by all CoRE endpoints is essential. This specification defines a format based on the CoRE Link-Format to represent binding

information along with the rules to define a binding method which is a specialized relationship between two resources.

The purpose of such a binding is to synchronize content updates between a source resource and a destination resource. The destination resource MAY be a group resource if the authority component of the destination URI contains a group address (either a multicast address or a name that resolves to a multicast address). Since a binding is unidirectional, the binding entry defining a relationship is present only on one endpoint. The binding entry may be located either on the source or the destination endpoint depending on the binding method.

Conditional Notification Attributes defined in Section 3 can be used with Link Bindings in order to customize the notification behavior and timing.

4.1. The "bind" attribute and Binding Methods

A binding method defines the rules to generate the network-transfer exchanges that synchronize state between source and destination resources. By using REST methods content is sent from the source resource to the destination resource.

This specification defines a new CoRE link attribute "bind". This is the identifier for a binding method which defines the rules to synchronize the destination resource. This attribute is mandatory.

Attribute	Parameter	Value
Binding method	bind	xs:string

Table 2: The bind attribute

The following table gives a summary of the binding methods defined in this specification.

Name	Identifier	Location	Method
Polling	poll	Destination	GET
Observe	obs	Destination	GET + Observe
Push	push	Source	PUT
Execute	exec	Source	POST

Table 3: Binding Method Summary

The description of a binding method defines the following aspects:

Identifier: This is the value of the "bind" attribute used to identify the method.

Location: This information indicates whether the binding entry is stored on the source or on the destination endpoint.

REST Method: This is the REST method used in the Request/Response exchanges.

Conditional Notification: How Conditional Notification Attributes are used in the binding.

The binding methods are described in more detail below.

4.1.1. Polling

The Polling method consists of sending periodic GET requests from the destination endpoint to the source resource and copying the content to the destination resource. The binding entry for this method MUST be stored on the destination endpoint. The destination endpoint MUST ensure that the polling frequency does not exceed the limits defined by the pmin and pmax attributes of the binding entry. The copying process MAY filter out content from the GET requests using value-based conditions (e.g based on the Change Step, Less Than, Greater Than attributes).

4.1.2. Observe

The Observe method creates an observation relationship between the destination endpoint and the source resource. On each notification the content from the source resource is copied to the destination resource. The creation of the observation relationship requires the

CoAP Observation mechanism [RFC7641] hence this method is only permitted when the resources are made available over CoAP. The binding entry for this method MUST be stored on the destination endpoint. The binding conditions are mapped as query parameters in the Observe request (see Section 3).

4.1.3. Push

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.

4.1.4. 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].

4.2. Link Relation

Since Binding involves the creation of a link between two resources, Web Linking and the CoRE Link-Format used to represent binding information. This involves the creation of a new relation type, "boundto". In a Web link with this relation type, the target URI contains the location of the source resource and the context URI points to the destination resource.

5. Binding Table

The Binding Table is a special resource that describes the bindings on an endpoint. An endpoint offering a representation of the Binding Table resource SHOULD indicate its presence and enable its discovery by advertising a link at "/.well-known/core" [RFC6690]. If so, the Binding Table resource MUST be discoverable by using the Resource Type (rt) 'core.bnd'.

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

Resource	rt=	Methods	Content-Format
Binding Table	core.bnd	GET, PUT	link-format

Table 4: Binding Table Description

The REST methods GET and PUT are used to manipulate a Binding Table. A GET request simply returns the current state of a Binding Table. A request with a PUT method and a content format of application/link-format is used to clear the bindings to the table or replaces its entire contents. All links in the payload of a PUT rquest 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 2: 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.

6. Implementation Considerations

When using multiple resource bindings (e.g. multiple Observations of resource) with different bands, consideration should be given to the resolution of the resource value when setting sequential bands. For example: Given BandA (Abmn=10, Bbmx=20) and BandB (Bbmn=21, Bbmx=30). If the resource value returns an integer then notifications for values between and inclusive of 10 and 30 will be triggered. Whereas if the resolution is to one decimal point (0.1) then notifications for values 20.1 to 20.9 will not be triggered.

The use of the notification band minimum and maximum allow for a synchronization whenever a change in the resource value occurs. Theoretically this could occur in-line with the server internal sample period for the determining the resource value. Implementors SHOULD consider the resolution needed before updating the resource, e.g. updating the resource when a temperature sensor value changes by 0.001 degree versus 1 degree.

The initiation of a Link Binding can be delegated from a client to a link state machine implementation, which can be an embedded client or a configuration tool. Implementation considerations have to be given to how to monitor transactions made by the configuration tool with regards to Link Bindings, as well as any errors that may arise with establishing Link Bindings in addition to established Link Bindings.

7. Security Considerations

Consideration has to be given to what kinds of security credentials the state machine of a configuration tool or an embedded client needs to be configured with, and what kinds of access control lists client implementations should possess, so that transactions on creating Link Bindings and handling error conditions can be processed by the state machine.

8. IANA Considerations

8.1. Resource Type value 'core.bnd'

This specification registers a new Resource Type Link Target Attribute 'core.bnd' in the Resource Type (rt=) registry established as per [RFC6690].

Attribute Value: core.bnd

Description: See Section 5. This attribute value is used to discover the resource representing a binding table, which describes the link bindings between source and destination resources for the purposes of synchronizing their content.

Reference: This specification. Note to RFC editor: please insert the RFC of this specification.

Notes: None

8.2. Link Relation Type

This specification registers the new "boundto" link relation type as per [RFC8288].

Relation Name: boundto

Description: The purpose of a boundto relation type is to indicate that there is a binding between a source resource and a destination resource for the purposes of synchronizing their content.

Reference: This specification. Note to RFC editor: please insert the RFC of this specification.

Notes: None

Application Data: None

9. Acknowledgements

Acknowledgement is given to colleagues from the SENSEI project who were critical in the initial development of the well-known REST interface concept, to members of the IPSO Alliance where further requirements for interface types have been discussed, and to Szymon Sasin, Cedric Chauvenet, Daniel Gavelle and Carsten Bormann who have provided useful discussion and input to the concepts in this specification. Christian Amsuss supplied a comprehensive review of draft -06. Hannes Tschofenig and Mert Ocak highlighted syntactical corrections in the usage of pmax and pmin in a query. Discussions with Ari Keraenen led to the addition of an extra binding method supporting POST operations.

10. Contributors

Matthieu Vial
Schneider-Electric
Grenoble
France

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

11. Changelog

draft-ietf-core-dynlink-10

- Binding methods now support both POST and PUT operations for server push.

draft-ietf-core-dynlink-09

- Corrections in Table 1, Table 2, Figure 2.
- Clarifications for additional operations to binding table added in section 5
- Additional examples in Appendix A

draft-ietf-core-dynlink-08

- Reorganize the draft to introduce Conditional Notification Attributes at the beginning
- Made pmin and pmax type xs:decimal to accommodate fractional second timing
- updated the attribute descriptions. lt and gt notify on all crossings, both directions
- updated Binding Table description, removed interface description but introduced core.bnd rt attribute value

draft-ietf-core-dynlink-07

- Added reference code to illustrate attribute interactions for observations

draft-ietf-core-dynlink-06

- Document restructure and refactoring into three main sections
- Clarifications on band usage
- Implementation considerations introduced
- Additional text on security considerations

draft-ietf-core-dynlink-05

- Addition of a band modifier for gt and lt, adapted from draft-groves-core-obsattr
- Removed statement prescribing gt MUST be greater than lt

draft-ietf-core-dynlink-03

- 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".
- New author and editor added.

draft-ietf-core-dynlink-02

- 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 conlict with the core resource directory draft lifetime "lt" attribute.

- Clause 6.1: Addressed the editor's note by changing the link target attribute to "core.binding".

- Added Appendix A for examples.

draft-ietf-core-dynlink-01

- General: The term state synchronization has been introduced to describe the process of synchronization between destination and source resources.

- General: The document has been restructured the make the information flow better.

- Clause 3.1: The descriptions of the binding attributes have been updated to clarify their usage.

- Clause 3.1: A new clause has been added to discuss the interactions between the resources.

- Clause 3.4: Has been simplified to refer to the descriptions in 3.1. As the text was largely duplicated.

- Clause 4.1: Added a clarification that individual resources may be removed from the binding table.

- Clause 6: Formailised the IANA considerations.

draft-ietf-core-dynlink Initial Version 00:

- This is a copy of draft-groves-core-dynlink-00

draft-groves-core-dynlink Draft Initial Version 00:

- This initial version is based on the text regarding the dynamic linking functionality in I.D.ietf-core-interfaces-05.

- The WADL description has been dropped in favour of a thorough textual description of the REST API.

12. References

12.1. Normative References

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

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

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

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

12.2. Informative References

[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-04 (work in progress), July 2019.

[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>.

Appendix A. Examples

This appendix provides some examples of the use of binding attribute / observe attributes.

Note: For brevity the only the method or response code is shown in the header field.

A.1. Minimum Period (pmin) example

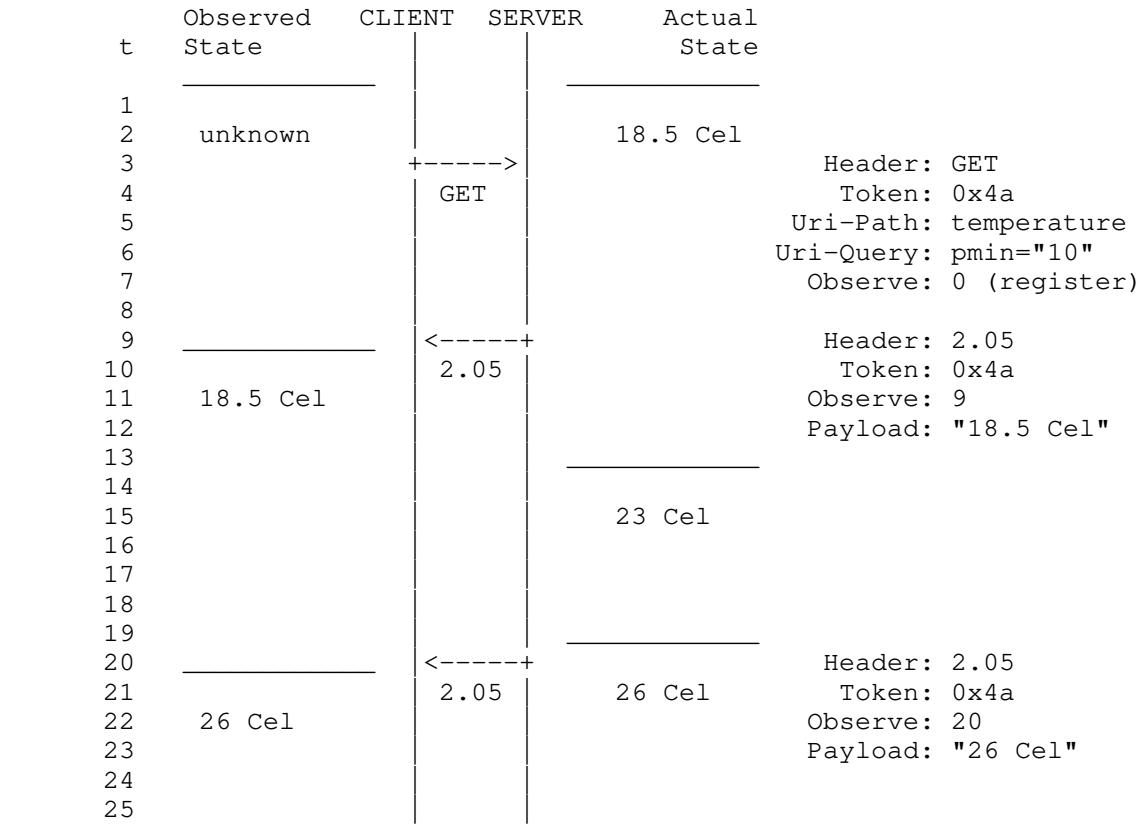
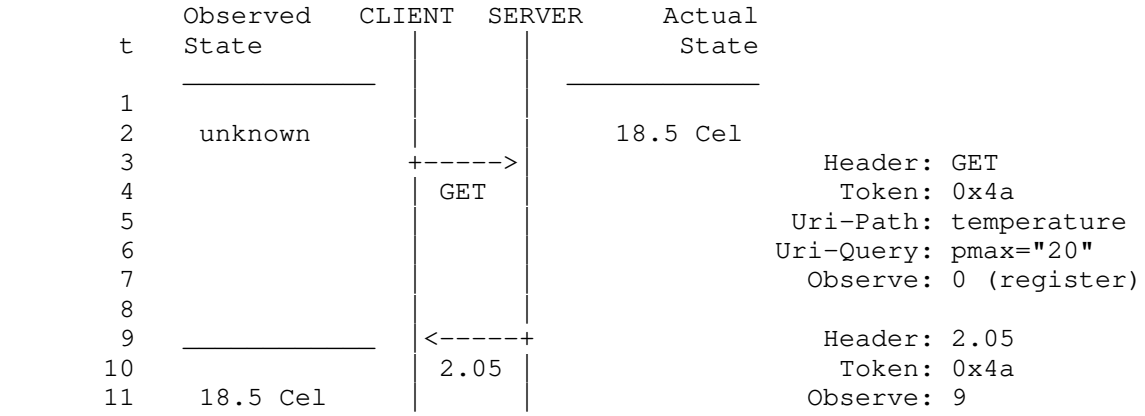


Figure 3: Client registers and receives one notification of the current state and one of a new state state when pmin time expires.

A.2. Maximum Period (pmax) example



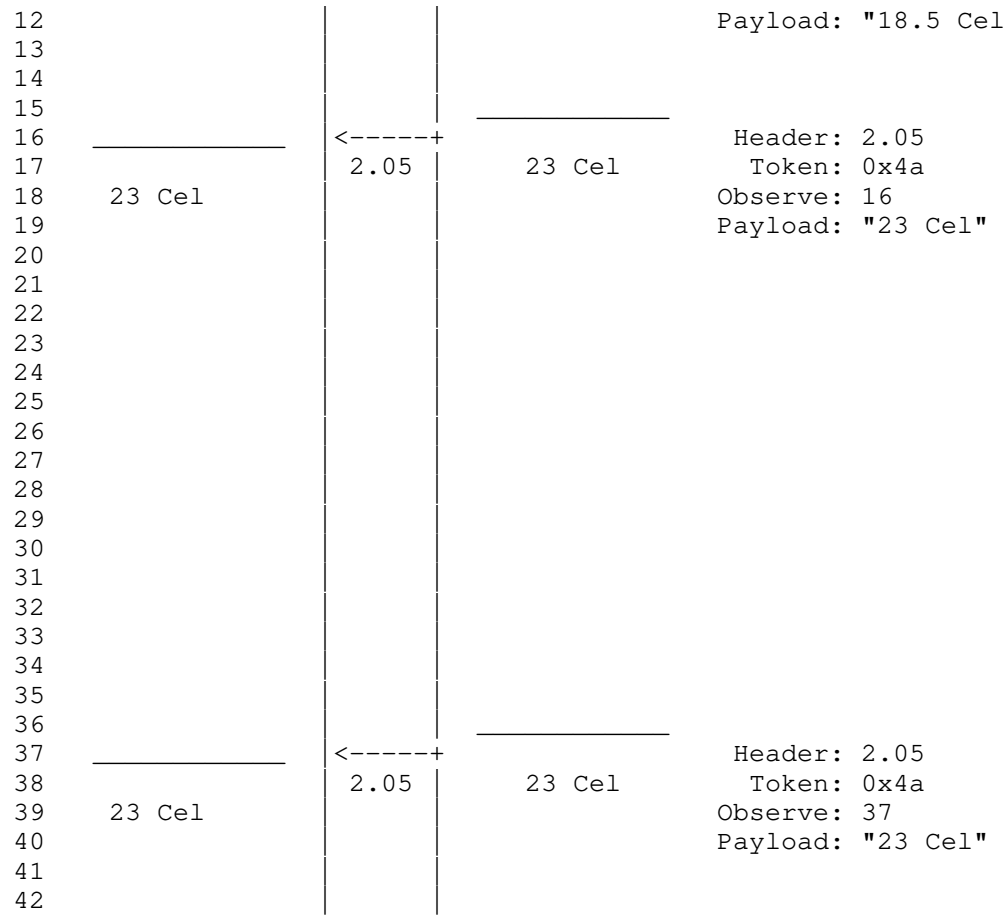


Figure 4: Client registers and receives one notification of the current state, one of a new state and one of an unchanged state when pmax time expires.

A.3. Greater Than (gt) example

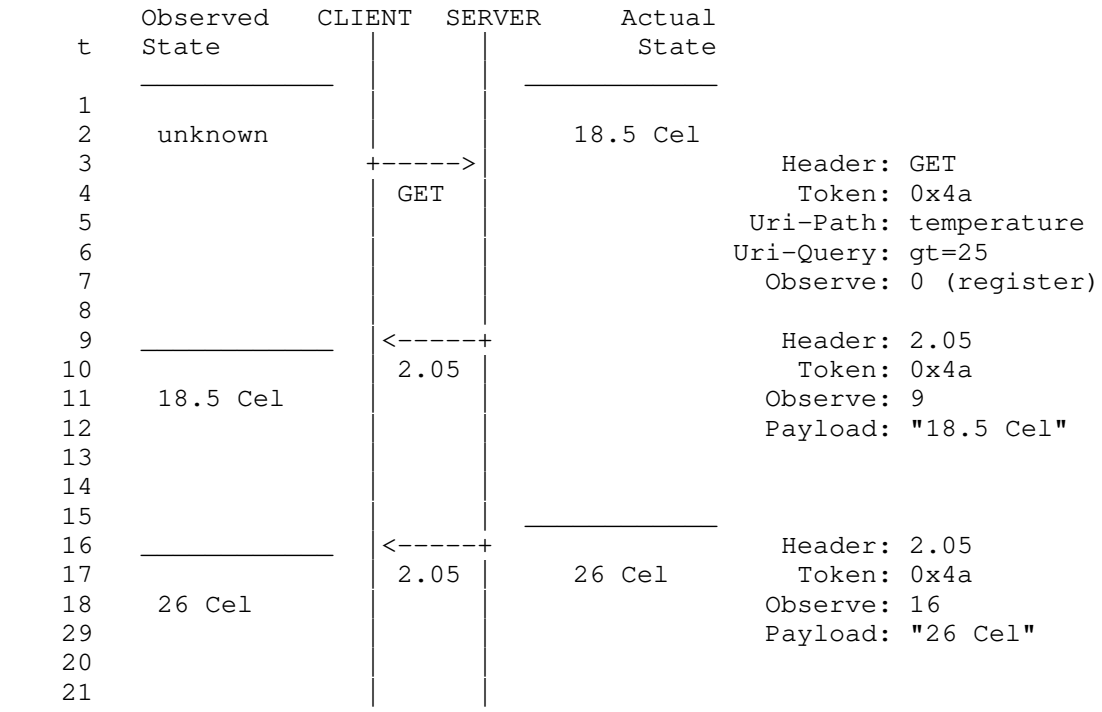
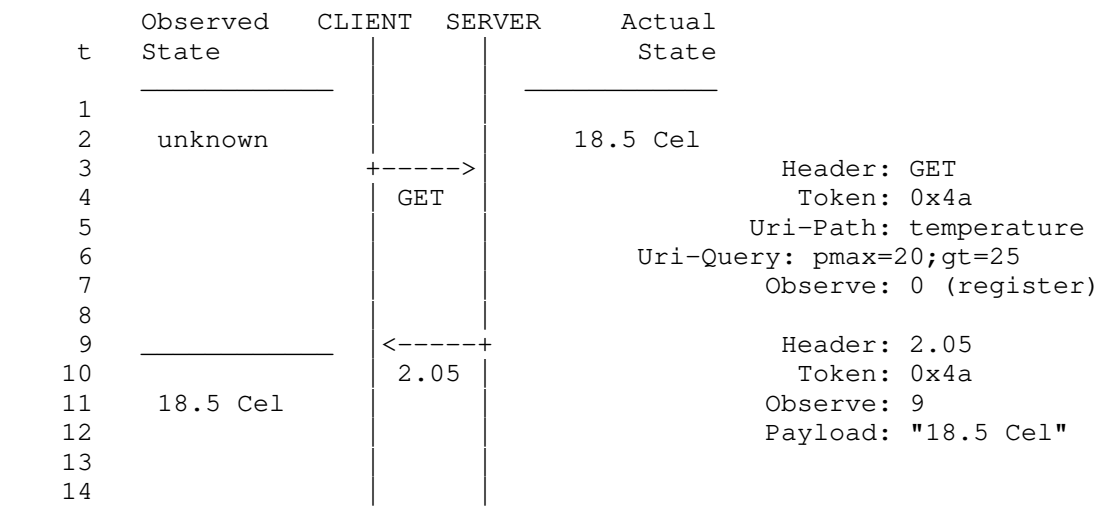


Figure 5: Client registers and receives one notification of the current state and one of a new state when it passes through the greater than threshold of 25.

A.4. Greater Than (gt) and Period Max (pmax) example



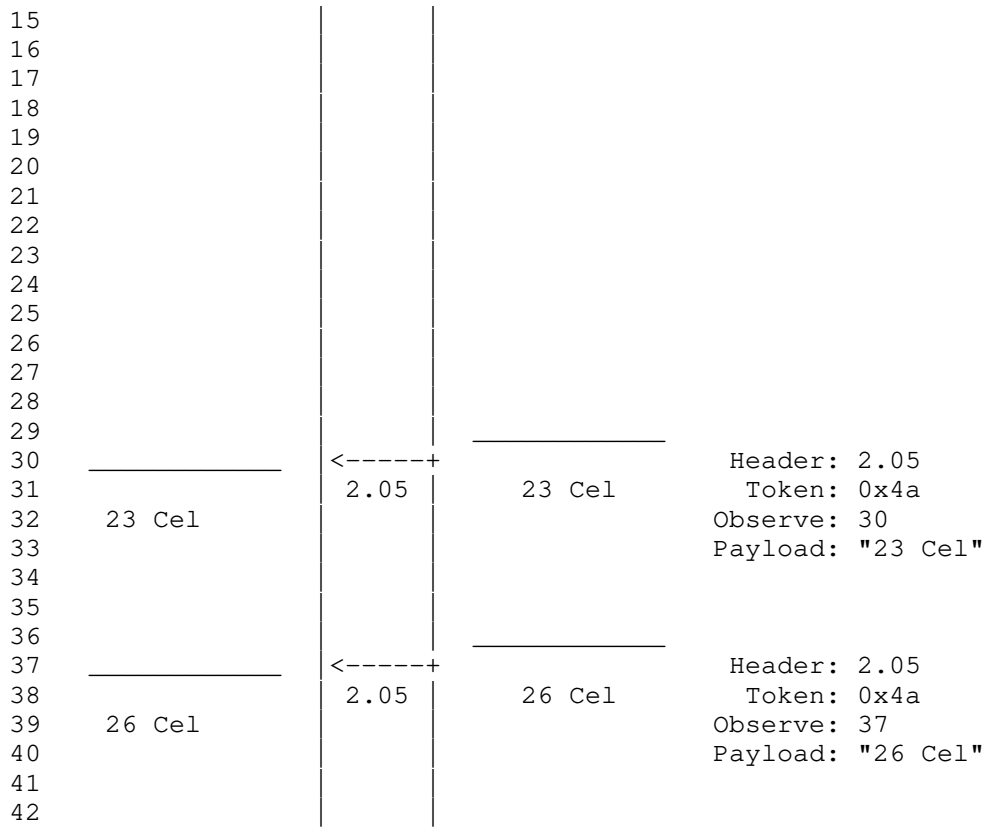


Figure 6: Client registers and receives one notification of the current state, one when pmax time expires and one of a new state when it passes through the greater than threshold of 25.

Authors' Addresses

Zach Shelby
ARM
Kidekuja 2
Vuokatti 88600
FINLAND

Phone: +358407796297
Email: zach.shelby@arm.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Email: michael.koster@smartthings.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: 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:

- The git repository for the draft is found at https://github.com/core-wg/interfaces

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

Table of Contents

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 contrained 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 Paramter 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 /l/ (Content-Format: application/link-format)
</s/light>,</s/temp>
Res: 2.04 Changed

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

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

Req: GET /l/
Res: 2.05 Content (application/senml+json)
[
   { "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 /l/
Res: 2.02 Deleted
```

4.4. Sensor

The Sensor interface allows the value of a sensor resource to be read (GET). The Media type of the resource can be either plain text or SenML. Plain text MAY be used for a single measurement that does not require meta-data. For a measurement with meta-data such as a unit or time stamp, SenML SHOULD be used. A resource with this interface MAY use SenML to return multiple measurements in the same representation, for example a list of recent measurements.

The following are examples of Sensor interface requests in both text/plain and application/senml+json.

```
Req: GET /s/humidity (Accept: text/plain)
Res: 2.05 Content (text/plain)
80

Req: GET /s/humidity (Accept: application/senml+json)
Res: 2.05 Content (application/senml+json)
[
    { "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:

- Version update, with changes in editor's contact information

Changes from -12 to -13:

- SenML examples now use the Base Name (bn) labels from RFC 8428
- Security considerations discusses client misuse of linked batches

Changes from -11 to -12:

- Removed all text referring to function sets/profiles
- Clarified list collections
- Content-formats for collections and items rectified
- Simplified Appendix A and removed Appendix B

Changes from -10 to -11:

- Added a new Section 3.4 for Link Embedding
- Updated examples in Section 3.5
- Removed "Service Discovery" from Terminologies
- Removed discussion of function sets

Changes from -09 to -10:

- Section 1: Amendments to remove discussing properties. *
- New author and editor added.

Changes from -08 to -09:

- Section 3.6: Modified to indicate that the entire collection resource is returned.
- General: Added editor's note with open issues.

Changes from -07 to -08:

- Section 3.3: Modified Accepts to Accept header option.
- Addressed the editor's note in Section 4.1 to clarify the use of the Accept option.

Changes from -06 to -07:

- Corrected Figure 1 sub-resource names e.g. tmp to temp and hum to humidity.
- Addressed the editor's note in Section 4.2.
- Removed section on function sets and profiles as agreed to at the IETF#97.

Changes from -05 to -06:

- Updated the abstract.
- Section 1: Updated introduction.
- Section 2: Alphabetised the order
- Section 2: Removed the collections definition in favour of the complete definition in the collections section.
- Removed section 3 on interfaces in favour of an updated definition in section 1.3.
- General: Changed interface type to interface description as that is the term defined in RFC6690.
- Removed section on future interfaces.

- Section 8: Updated IANA considerations.
- Added Appendix A to discuss current state of the art wrt to collections, function sets etc.

Changes from -04 to -05:

- Removed Link Bindings and Observe attributes. This functionality is now contained in I-D.ietf-core-dynlink.
- Hypermedia collections have been removed. This is covered in a new T2TRG draft.
- The WADL description has been removed.
- Fixed minor typos.
- Updated references.

Changes from -03 to -04:

- Fixed tickets #385 and #386.
- Changed abstract and into to better describe content.
- Focus on Interface and not function set/profiles in intro.
- Changed references from draft-core-observe to RFC7641.
- Moved Function sets and Profiles to section after Interfaces.
- Moved Observe Attributes to the Link Binding section.
- Add a Collection section to describe the collection types.
- Add the Hypermedia Collection Interface Description.

Changes from -02 to -03:

- Added lt and gt to binding format section.
- Added pmin and pmax observe parameters to Observation Attributes.
- Changed the definition of lt and gt to limit crossing.
- Added definitions for getattr and setattr to WADL.
- Added getattr and setattr to observable interfaces.

- Removed query parameters from Observe definition.
- Added observe-cancel definition to WADL and to observable interfaces.

Changes from -01 to -02:

- Updated the date and version, fixed references.
- "Removed pmin and pmax observe parameters "[Ticket #336]"."

Changes from -00 to WG Document -01

- Improvements to the Function Set section.

Changes from -05 to WG Document -00

- Updated the date and version.

Changes from -04 to -05

- 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

- Draft refresh

Changes from -02 to -03

- Added Bindings
- Updated all rt= and if= for the new Link Format IANA rules

Changes from -01 to -02

- Defined a Function Set and its guidelines.
- Added the Link List interface.
- Added the Linked Batch interface.
- Improved the WADL interface definition.
- 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@smartthings.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.

Table of Contents

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:

- 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
- 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:

- the collection of Web links to a JSON or CBOR array of links;
- 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 unqouted 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:

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

- 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)
         646573637269626564279    # "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?
```

```
          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].

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is a web transfer protocol, designed for constrained nodes and networks [RFC7228], and may be mapped from HTTP [RFC8075]. CoAP specifies the use of proxies for scalability and efficiency and references DTLS [RFC6347] for security. CoAP-to-CoAP, HTTP-to-CoAP, and CoAP-to-HTTP proxies require DTLS or TLS [RFC8446] to be terminated at the proxy. The proxy therefore not only has access to the data required for performing the intended proxy functionality, but is also able to eavesdrop on, or manipulate any part of, the message payload and metadata in transit between the endpoints. The proxy can also inject, delete, or reorder packets since they are no longer protected by (D)TLS.

This document defines the Object Security for Constrained RESTful Environments (OSCORE) security protocol, protecting CoAP and CoAP-mappable HTTP requests and responses end-to-end across intermediary nodes such as CoAP forward proxies and cross-protocol translators including HTTP-to-CoAP proxies [RFC8075]. In addition to the core CoAP features defined in [RFC7252], OSCORE supports the Observe [RFC7641], Block-wise [RFC7959], and No-Response [RFC7967] options, as well as the PATCH and FETCH methods [RFC8132]. An analysis of end-to-end security for CoAP messages through some types of intermediary nodes is performed in [I-D.hartke-core-e2e-security-reqs]. OSCORE essentially protects the RESTful interactions; the request method, the requested resource, the message payload, etc. (see Section 4). OSCORE protects neither the CoAP Messaging Layer nor the CoAP Token which may change between the endpoints, and those are therefore processed as defined in [RFC7252]. Additionally, since the message formats for CoAP over unreliable transport [RFC7252] and for CoAP over reliable transport [RFC8323] differ only in terms of CoAP Messaging Layer, OSCORE can be applied to both unreliable and reliable transports (see Figure 1).

OSCORE works in very constrained nodes and networks, thanks to its small message size and the restricted code and memory requirements in addition to what is required by CoAP. Examples of the use of OSCORE are given in Appendix A. OSCORE may be used over any underlying layer, such as e.g. UDP or TCP, and with non-IP transports (e.g.,

[I-D.bormann-6lo-coap-802-15-ie]). OSCORE may also be used in different ways with HTTP. OSCORE messages may be transported in HTTP, and OSCORE may also be used to protect CoAP-mappable HTTP messages, as described below.

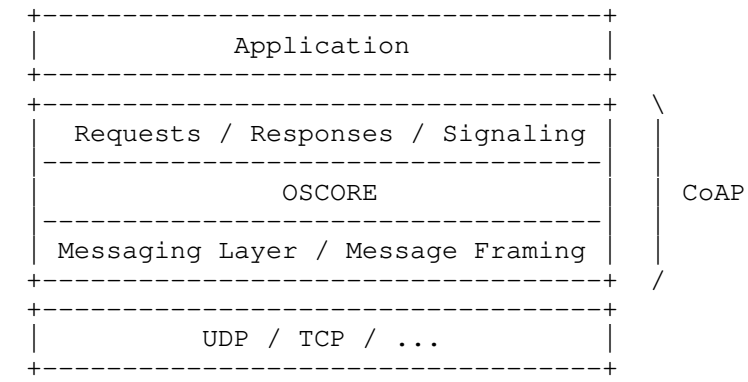


Figure 1: Abstract Layering of CoAP with OSCORE

OSCORE is designed to protect as much information as possible while still allowing CoAP proxy operations (Section 10). It works with existing CoAP-to-CoAP forward proxies [RFC7252], but an OSCORE-aware proxy will be more efficient. HTTP-to-CoAP proxies [RFC8075] and CoAP-to-HTTP proxies can also be used with OSCORE, as specified in Section 11. OSCORE may be used together with TLS or DTLS over one or more hops in the end-to-end path, e.g. transported with HTTPS in one hop and with plain CoAP in another hop. The use of OSCORE does not affect the URI scheme and OSCORE can therefore be used with any URI scheme defined for CoAP or HTTP. The application decides the conditions for which OSCORE is required.

OSCORE uses pre-shared keys which may have been established out-of-band or with a key establishment protocol (see Section 3.2). The technical solution builds on CBOR Object Signing and Encryption (COSE) [RFC8152], providing end-to-end encryption, integrity, replay protection, and binding of response to request. A compressed version of COSE is used, as specified in Section 6. The use of OSCORE is signaled in CoAP with a new option (Section 2), and in HTTP with a new header field (Section 11.1) and content type (Section 13.5). The solution transforms a CoAP/HTTP message into an "OSCORE message" before sending, and vice versa after receiving. The OSCORE message is a CoAP/HTTP message related to the original message in the following way: the original CoAP/HTTP message is translated to CoAP (if not already in CoAP) and protected in a COSE object. The encrypted message fields of this COSE object are transported in the CoAP payload/HTTP body of the OSCORE message, and the OSCORE option/

header field is included in the message. A sketch of an exchange of OSCORE messages, in the case of the original message being CoAP, is provided in Figure 2. The use of OSCORE with HTTP is detailed in Section 11.

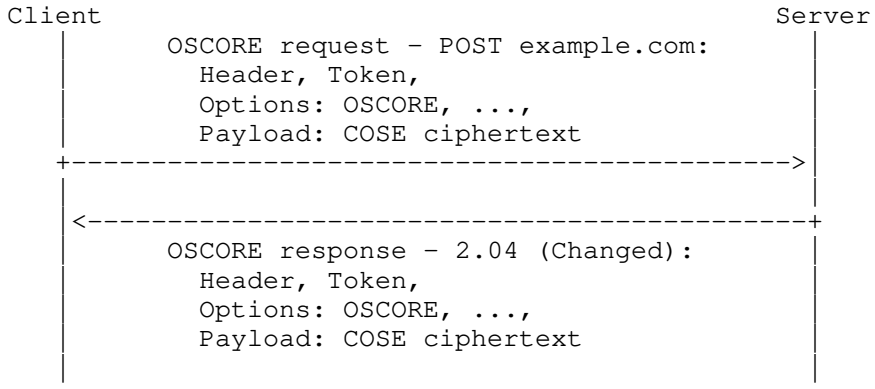


Figure 2: Sketch of CoAP with OSCORE

An implementation supporting this specification MAY implement only the client part, MAY implement only the server part, or MAY implement only one of the proxy parts.

1.1. Terminology

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

Readers are expected to be familiar with the terms and concepts described in CoAP [RFC7252], Observe [RFC7641], Block-wise [RFC7959], COSE [RFC8152], CBOR [RFC7049], CDDL [I-D.ietf-cbor-cddl] as summarized in Appendix E, and constrained environments [RFC7228].

The term "hop" is used to denote a particular leg in the end-to-end path. The concept "hop-by-hop" (as in "hop-by-hop encryption" or "hop-by-hop fragmentation") opposed to "end-to-end", is used in this document to indicate that the messages are processed accordingly in the intermediaries, rather than just forwarded to the next node.

The term "stop processing" is used throughout the document to denote that the message is not passed up to the CoAP Request/Response Layer (see Figure 1).

The terms Common/Sender/Recipient Context, Master Secret/Salt, Sender ID/Key, Recipient ID/Key, ID Context, and Common IV are defined in Section 3.1.

2. The OSCORE Option

The OSCORE option defined in this section (see Figure 3, which extends Table 4: Options of [RFC7252]) indicates that the CoAP message is an OSCORE message and that it contains a compressed COSE object (see Sections 5 and 6). The OSCORE option is critical, safe to forward, part of the cache key, and not repeatable.

No.	C	U	N	R	Name	Format	Length	Default
TBD1	x				OSCORE	(*)	0-255	(none)

C = Critical, U = Unsafe, N = NoCacheKey, R = Repeatable
(*) See below.

Figure 3: The OSCORE Option

The OSCORE option includes the OSCORE flag bits (Section 6), the Sender Sequence Number, the Sender ID, and the ID Context when these fields are present (Section 3). The detailed format and length is specified in Section 6. If the OSCORE flag bits are all zero (0x00) the Option value SHALL be empty (Option Length = 0). An endpoint receiving a CoAP message without payload, that also contains an OSCORE option SHALL treat it as malformed and reject it.

A successful response to a request with the OSCORE option SHALL contain the OSCORE option. Whether error responses contain the OSCORE option depends on the error type (see Section 8).

For CoAP proxy operations, see Section 10.

3. The Security Context

OSCORE requires that client and server establish a shared security context used to process the COSE objects. OSCORE uses COSE with an Authenticated Encryption with Additional Data (AEAD, [RFC5116]) algorithm for protecting message data between a client and a server. In this section, we define the security context and how it is derived in client and server based on a shared secret and a key derivation function.

3.1. Security Context Definition

The security context is the set of information elements necessary to carry out the cryptographic operations in OSCORE. For each endpoint, the security context is composed of a "Common Context", a "Sender Context", and a "Recipient Context".

The endpoints protect messages to send using the Sender Context and verify messages received using the Recipient Context, both contexts being derived from the Common Context and other data. Clients and servers need to be able to retrieve the correct security context to use.

An endpoint uses its Sender ID (SID) to derive its Sender Context, and the other endpoint uses the same ID, now called Recipient ID (RID), to derive its Recipient Context. In communication between two endpoints, the Sender Context of one endpoint matches the Recipient Context of the other endpoint, and vice versa. Thus, the two security contexts identified by the same IDs in the two endpoints are not the same, but they are partly mirrored. Retrieval and use of the security context are shown in Figure 4.

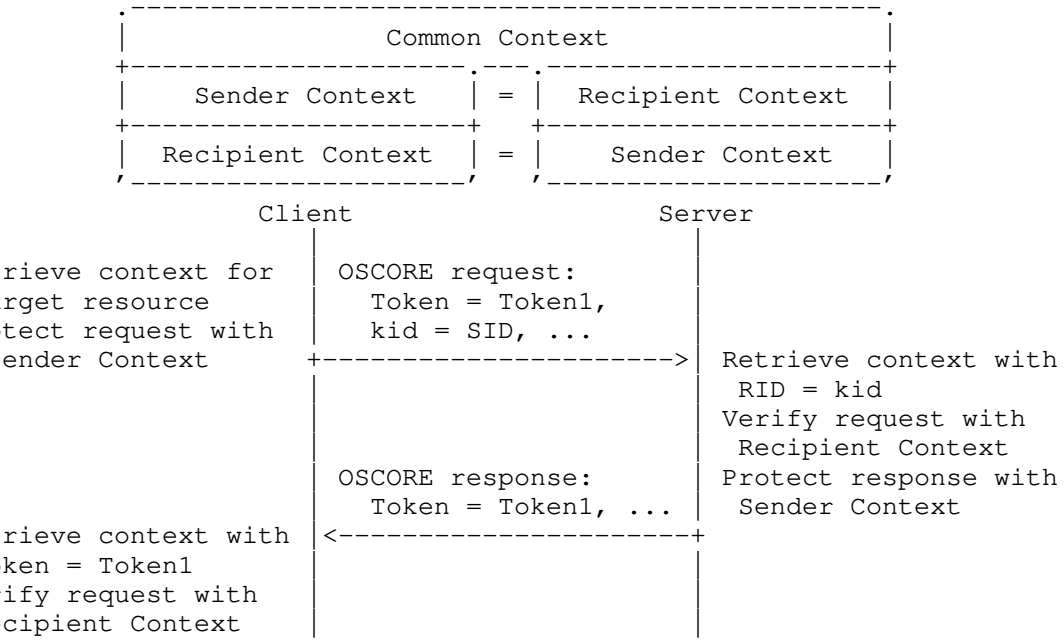


Figure 4: Retrieval and Use of the Security Context

The Common Context contains the following parameters:

- AEAD Algorithm. The COSE AEAD algorithm to use for encryption.

- HKDF Algorithm. An HMAC-based key derivation function HKDF [RFC5869] used to derive Sender Key, Recipient Key, and Common IV.

- Master Secret. Variable length, random byte string (see Section 12.3) used to derive AEAD keys and Common IV.

- Master Salt. Optional variable length byte string containing the salt used to derive AEAD keys and Common IV.

- 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.

- 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:

- 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.

- 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.

- 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:

- 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.

- 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.

- 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:

- Master Secret
- Sender ID
- 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:

- AEAD Algorithm
  - Default is AES-CCM-16-64-128 (COSE algorithm encoding: 10)
- Master Salt
  - Default is the empty byte string
- HKDF Algorithm
  - Default is HKDF SHA-256
- 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:

- salt is the Master Salt as defined above
- IKM is the Master Secret as defined above
- 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:

- 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.
- id_context is the ID Context, or nil if ID Context is not provided.
- alg_aead is the AEAD Algorithm, encoded as defined in [RFC8152].

- type is "Key" or "IV". The label is an ASCII string, and does not include a trailing NUL byte.

- 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:

- Class E: encrypted and integrity protected,
- Class I: integrity protected only, or
- 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. 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:

- Proxy-Uri = "coap://example.com/resource?q=1"

During OSCORE processing, Proxy-Uri is split into:

- Proxy-Scheme = "coap"
- Uri-Host = "example.com"
- Uri-Port = "5683"
- Uri-Path = "resource"
- 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:

- Uri-Path = "resource"
- 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:

- 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.

- 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].

- 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.

- 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.

- 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:

- To comply with [RFC8323], an initial empty CSM message SHALL be sent. The subsequent signaling message SHALL be protected.

- 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.

- 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).

- 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

- The 'protected' field is empty.

- 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.

- 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:

- If the client has a unique identifier in some namespace then that identifier can be used as ID Context.

- The ID Context may be used to add randomness into new Sender and Recipient Contexts, see Appendix B.2.

- 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.mcgrew-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.

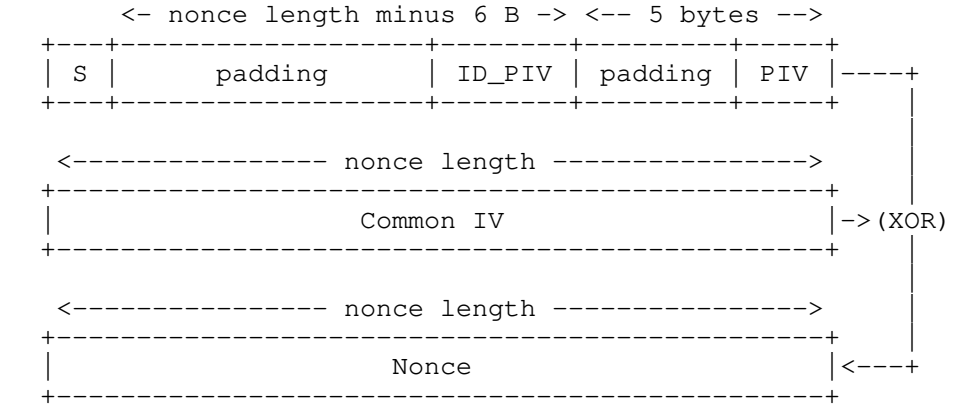


Figure 8: AEAD Nonce Formation

5.3. Plaintext

The plaintext is formatted as a CoAP message without Header (see Figure 9) consisting of:

- the Code of the original CoAP message as defined in Section 3 of [RFC7252]; and
- 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
- 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.

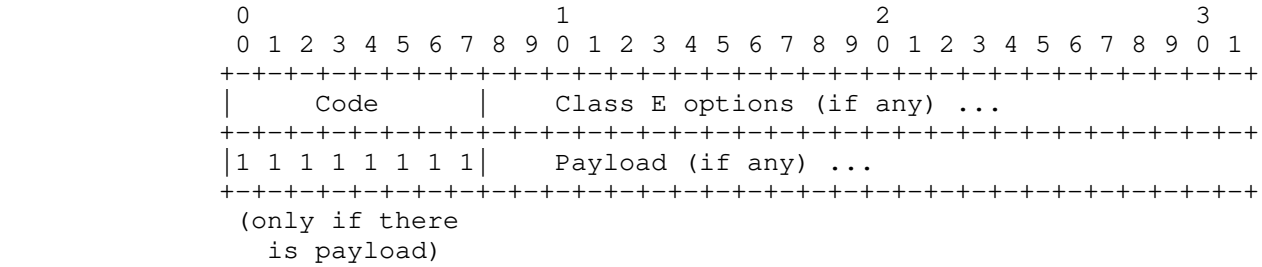


Figure 9: Plaintext

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:

- oscore_version: contains the OSCORE version number. Implementations of this specification MUST set this field to 1. Other values are reserved for future versions.

- algorithms: contains (for extensibility) an array of algorithms, according to this specification only containing alg_aead.

- alg_aead: contains the AEAD Algorithm from the security context used for the exchange (see Section 3.1).

- request_kid: contains the value of the 'kid' in the COSE object of the request (see Section 5).

- request_piv: contains the value of the 'Partial IV' in the COSE object of the request (see Section 5).

- 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:

- oscore_version: 0x01 (1 byte)
- algorithms: 0x810a (2 bytes)
- request_kid: 0x00 (1 byte)
- request_piv: 0x25 (1 byte)
- options: 0x (0 bytes)
- aad_array: 0x8501810a4100412540 (9 bytes)
- external_aad: 0x498501810a4100412540 (10 bytes)
- 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:

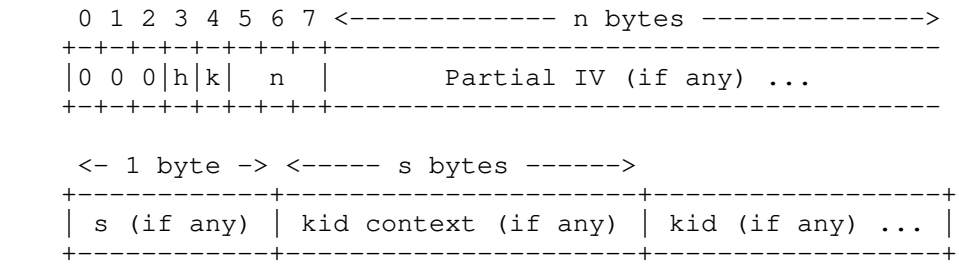


Figure 10: The OSCORE Option Value

- 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.

- The following n bytes encode the value of the Partial IV, if the Partial IV is present (n > 0).

- The following 1 byte encode the length of the 'kid context' (Section 5.1) s, if the 'kid context' flag is set (h = 1).

- The following s bytes encode the 'kid context', if the 'kid context' flag is set (h = 1).

- 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.mcgrew-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:

- 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.
- 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:

- 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.

- 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:

- 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.

- 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:

- SHALL bypass caching for the request if the OSCORE option is present
- 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:

- SHALL NOT initiate an Observe registration
- 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:

- the CoAP-to-HTTP proxy follows the redirect, instead of the CoAP client as in the HTTP case
- the CoAP-to-HTTP proxy copies the HTTP OSCORE header field and body to the new request
- 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:

- 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].
- 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:

- 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].
- 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 end-point 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 is 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":

- Name: kid context
- Label: TBD2
- Value Type: bstr
- Value Registry:
- Description: Identifies the context for 'kid'
- 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:

- 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.

- 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.

- description: This contains a brief description of the use of the bit.

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

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

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.

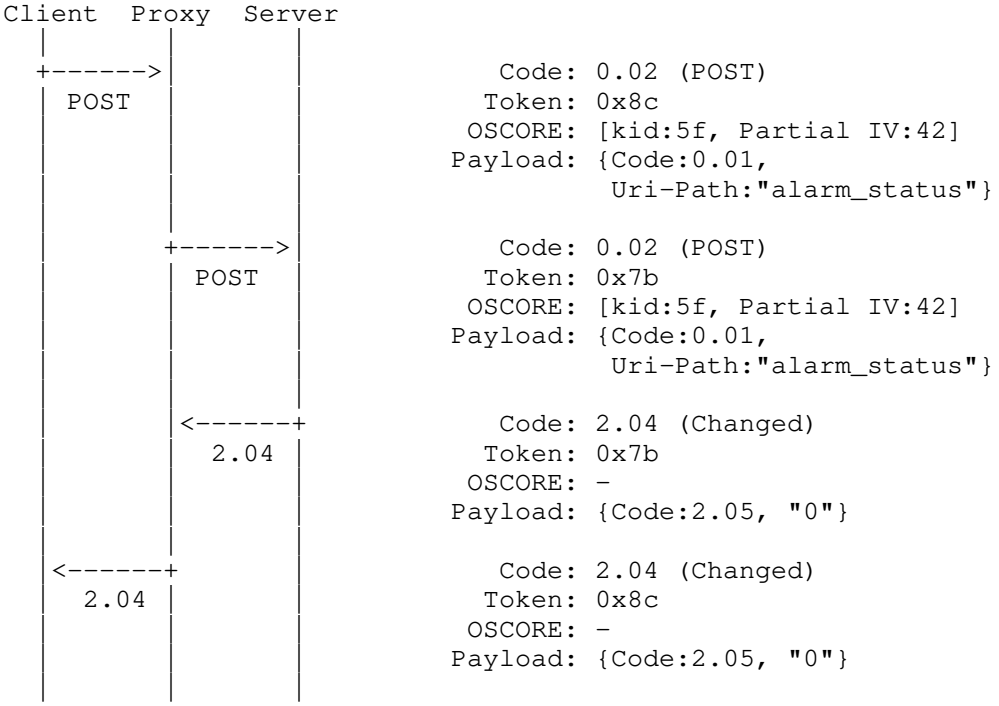


Figure 12: Secure Access to Sensor. Square brackets [...] indicate content of compressed COSE object. Curly brackets { ... } indicate encrypted data.

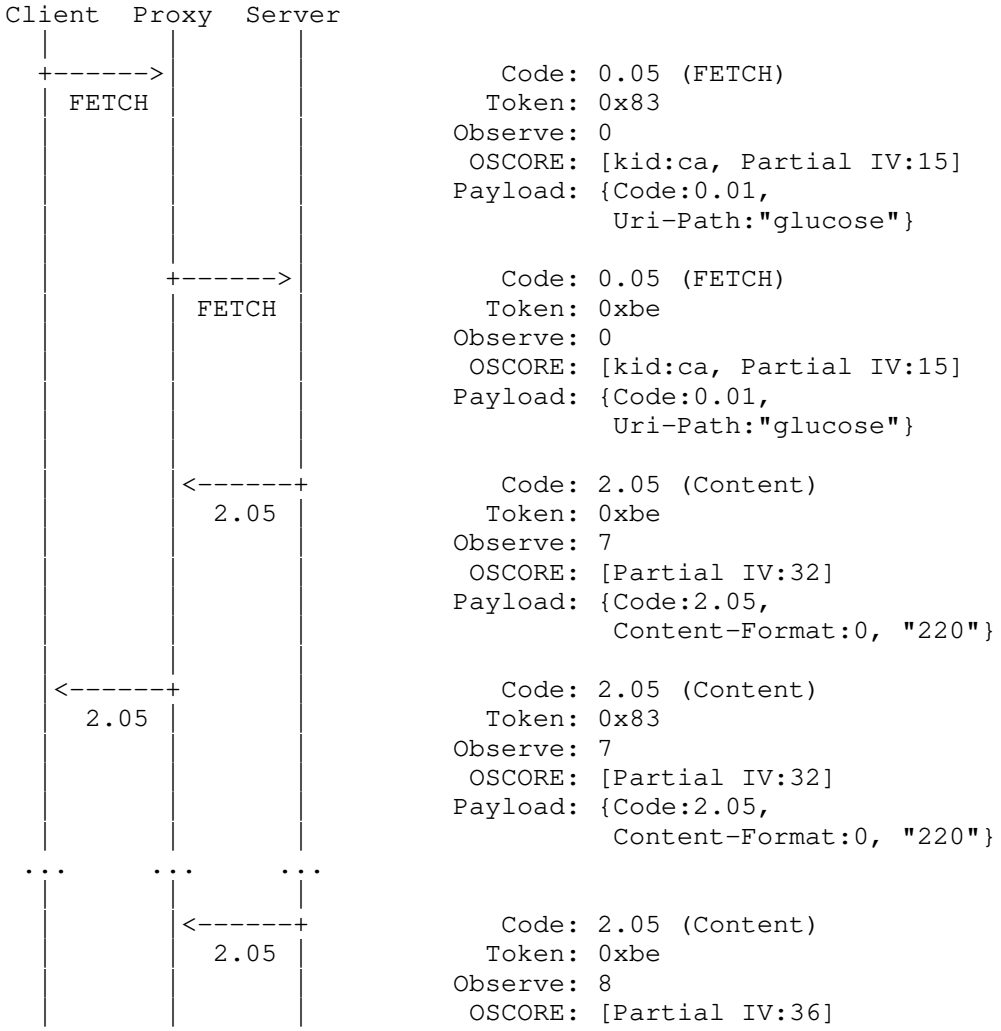
The request/response Codes are encrypted by OSCORE and only dummy Codes (POST/Changed) are visible in the header of the OSCORE message. The option Uri-Path ("alarm_status") and payload ("0") are encrypted.

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.



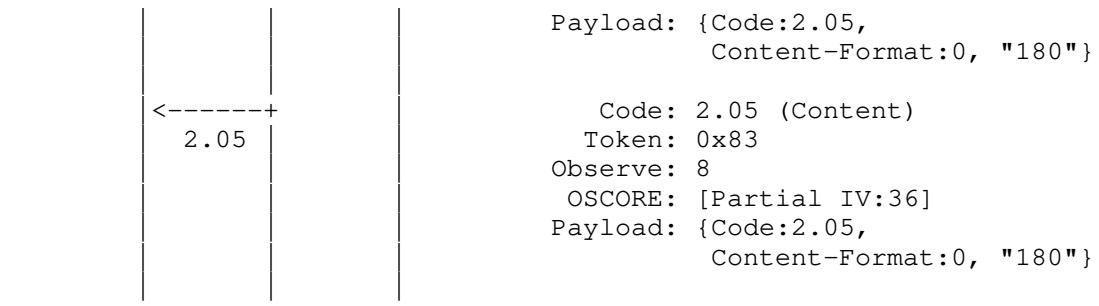


Figure 13: Secure Subscribe to Sensor. Square brackets [...] indicate content of compressed COSE object header. Curly brackets { ... } indicate encrypted data.

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. 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:

- 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.1.2. 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:

- 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).

- 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:

- 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.

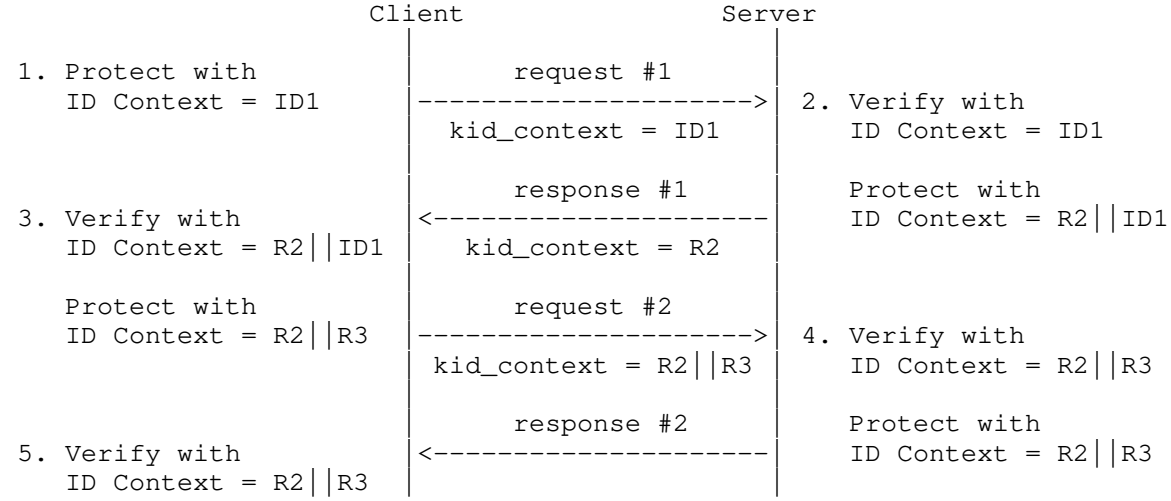


Figure 14: Protocol for establishing a new security context.

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 reponse #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:

- 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 || 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.

- 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 accidently 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:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- Master Salt: 0x9e7ca92223786340 (8 bytes)
- Sender ID: 0x (0 byte)
- Recipient ID: 0x01 (1 byte)

From the previous parameters,

- info (for Sender Key): 0x8540f60a634b657910 (9 bytes)
- info (for Recipient Key): 0x854101f60a634b657910 (10 bytes)
- info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- Sender Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- Recipient Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- sender nonce: 0x4622d4dd6d944168eefb54987c (13 bytes)
- recipient nonce: 0x4722d4dd6d944169eefb54987c (13 bytes)

C.1.2. Server

Inputs:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- Master Salt: 0x9e7ca92223786340 (8 bytes)

- Sender ID: 0x01 (1 byte)
- Recipient ID: 0x (0 byte)

From the previous parameters,

- info (for Sender Key): 0x854101f60a634b657910 (10 bytes)
- info (for Recipient Key): 0x8540f60a634b657910 (9 bytes)
- info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- Recipient Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- sender nonce: 0x4722d4dd6d944169eefb54987c (13 bytes)
- 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:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- Sender ID: 0x00 (1 byte)
- Recipient ID: 0x01 (1 byte)

From the previous parameters,

- info (for Sender Key): 0x854100f60a634b657910 (10 bytes)
- info (for Recipient Key): 0x854101f60a634b657910 (10 bytes)

- info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- Sender Key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- Recipient Key: 0xe57b5635815177cd679ab4bcec9d7dda (16 bytes)
- Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- sender nonce: 0xbf35ae297d2dace910c52e99f9 (13 bytes)
- recipient nonce: 0xbf35ae297d2dace810c52e99f9 (13 bytes)

C.2.2. Server

Inputs:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- Sender ID: 0x01 (1 byte)
- Recipient ID: 0x00 (1 byte)

From the previous parameters,

- info (for Sender Key): 0x854101f60a634b657910 (10 bytes)
- info (for Recipient Key): 0x854100f60a634b657910 (10 bytes)
- info (for Common IV): 0x8540f60a6249560d (8 bytes)

Outputs:

- Sender Key: 0xe57b5635815177cd679ab4bcec9d7dda (16 bytes)
- Recipient Key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- sender nonce: 0xbf35ae297d2dace810c52e99f9 (13 bytes)

- 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:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- Master Salt: 0x9e7ca92223786340 (8 bytes)
- Sender ID: 0x (0 byte)
- Recipient ID: 0x01 (1 byte)
- ID Context: 0x37cbf3210017a2d3 (8 bytes)

From the previous parameters,

- info (for Sender Key): 0x85404837cbf3210017a2d30a634b657910 (17 bytes)
- info (for Recipient Key): 0x8541014837cbf3210017a2d30a634b657910 (18 bytes)
- info (for Common IV): 0x85404837cbf3210017a2d30a6249560d (16 bytes)

Outputs:

- Sender Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- Recipient Key: 0xe39a0c7c77b43f03b4b39ab9a268699f (16 bytes)
- Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- sender nonce: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)
- recipient nonce: 0x2da58fb85ff1b81d0b7181b85e (13 bytes)

C.3.2. Server

Inputs:

- Master Secret: 0x0102030405060708090a0b0c0d0e0f10 (16 bytes)
- Master Salt: 0x9e7ca92223786340 (8 bytes)
- Sender ID: 0x01 (1 byte)
- Recipient ID: 0x (0 byte)
- ID Context: 0x37cbf3210017a2d3 (8 bytes)

From the previous parameters,

- info (for Sender Key): 0x8541014837cbf3210017a2d30a634b657910 (18 bytes)
- info (for Recipient Key): 0x85404837cbf3210017a2d30a634b657910 (17 bytes)
- info (for Common IV): 0x85404837cbf3210017a2d30a6249560d (16 bytes)

Outputs:

- Sender Key: 0xe39a0c7c77b43f03b4b39ab9a268699f (16 bytes)
- Recipient Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)

From the previous parameters and a Partial IV equal to 0 (both for sender and recipient):

- sender nonce: 0x2da58fb85ff1b81d0b7181b85e (13 bytes)
- 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:

- AEAD Algorithm: 10 (AES-CCM-16-64-128)
- Key Derivation Function: HKDF SHA-256
- Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

Sender Context:

- Sender ID: 0x (0 byte)
- Sender Key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- Partial IV: 0x14 (1 byte)
- kid: 0x (0 byte)
- external_aad: 0x8501810a40411440 (8 bytes)
- AAD: 0x8368456e6372797074304048850181 0a40411440 (20 bytes)
- plaintext: 0x01b3747631 (5 bytes)
- encryption key: 0xf0910ed7295e6ad4b54fc793154302ff (16 bytes)
- nonce: 0x4622d4dd6d944168eefb549868 (13 bytes)

From the previous parameter, the following is derived:

- OSCORE option value: 0x0914 (2 bytes)
- ciphertext: 0x612f1092f1776f1c1668b3825e (13 bytes)

From there:

- Protected CoAP request (OSCORE message): 0x44025d1f00003974396c6f6 3616c686f7374620914ff612f1092f1776f1c1668b3825e (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:

- AEAD Algorithm: 10 (AES-CCM-16-64-128)
- Key Derivation Function: HKDF SHA-256
- Common IV: 0xbe35ae297d2dace910c52e99f9 (13 bytes)

Sender Context:

- Sender ID: 0x00 (1 bytes)
- Sender Key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- Partial IV: 0x14 (1 byte)
- kid: 0x00 (1 byte)
- external_aad: 0x8501810a4100411440 (9 bytes)
- AAD: 0x8368456e6372797074304049850181 0a4100411440 (21 bytes)
- plaintext: 0x01b3747631 (5 bytes)
- encryption key: 0x321b26943253c7ffb6003b0b64d74041 (16 bytes)
- nonce: 0xbf35ae297d2dace910c52e99ed (13 bytes)

From the previous parameter, the following is derived:

- OSCORE option value: 0x091400 (3 bytes)
- ciphertext: 0x4ed339a5a379b0b8bc731fffb0 (13 bytes)

From there:

- Protected CoAP request (OSCORE message): 0x440271c30000b932396c6f6 3616c686f737463091400ff4ed339a5a379b0b8bc731fffb0 (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:

- AEAD Algorithm: 10 (AES-CCM-16-64-128)
- Key Derivation Function: HKDF SHA-256
- Common IV: 0x2ca58fb85ff1b81c0b7181b85e (13 bytes)
- ID Context: 0x37cbf3210017a2d3 (8 bytes)

Sender Context:

- Sender ID: 0x (0 bytes)
- Sender Key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- Sender Sequence Number: 20

The following COSE and cryptographic parameters are derived:

- Partial IV: 0x14 (1 byte)
- kid: 0x (0 byte)
- kid context: 0x37cbf3210017a2d3 (8 bytes)
- external_aad: 0x8501810a40411440 (8 bytes)
- AAD: 0x8368456e6372797074304048850181 0a40411440 (20 bytes)
- plaintext: 0x01b3747631 (5 bytes)
- encryption key: 0xaf2a1300a5e95788b356336eeecd2b92 (16 bytes)
- nonce: 0x2ca58fb85ff1b81c0b7181b84a (13 bytes)

From the previous parameter, the following is derived:

- OSCORE option value: 0x19140837cbf3210017a2d3 (11 bytes)

- ciphertext: 0x72cd7273fd331ac45cffbe55c3 (13 bytes)

From there:

- 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:

- AEAD Algorithm: 10 (AES-CCM-16-64-128)

- Key Derivation Function: HKDF SHA-256

- Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

Sender Context:

- Sender ID: 0x01 (1 byte)

- Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)

- Sender Sequence Number: 0

The following COSE and cryptographic parameters are derived:

- external_aad: 0x8501810a40411440 (8 bytes)

- AAD: 0x8368456e6372797074304048850181 0a40411440 (20 bytes)

- plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)

- encryption key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)

- nonce: 0x4622d4dd6d944168eefb549868 (13 bytes)

From the previous parameter, the following is derived:

- OSCORE option value: 0x (0 bytes)

- ciphertext: 0xdbaad1e9a7e7b2a813d3c31524378303cdafae119106 (22 bytes)

From there:

- 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:

- AEAD Algorithm: 10 (AES-CCM-16-64-128)

- Key Derivation Function: HKDF SHA-256

- Common IV: 0x4622d4dd6d944168eefb54987c (13 bytes)

Sender Context:

- Sender ID: 0x01 (1 byte)

- Sender Key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)

- Sender Sequence Number: 0

The following COSE and cryptographic parameters are derived:

- Partial IV: 0x00 (1 byte)

- external_aad: 0x8501810a40411440 (8 bytes)

- AAD: 0x8368456e6372797074304048850181 0a40411440 (20 bytes)
- plaintext: 0x45ff48656c6c6f20576f726c6421 (14 bytes)
- encryption key: 0xffb14e093c94c9cac9471648b4f98710 (16 bytes)
- nonce: 0x4722d4dd6d944169eefb54987c (13 bytes)

From the previous parameter, the following is derived:

- OSCORE option value: 0x0100 (2 bytes)
- ciphertext: 0x4d4c13669384b67354b2b6175ff4b8658c666a6cf88e (22 bytes)

From there:

- Protected CoAP response (OSCORE message): 0x64445d1f00003974920100 ff4d4c13669384b67354b2b6175ff4b8658c666a6cf88e (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:

- 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).

- 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.

- Request-response binding: An attacker should not be able to make a client match a response to the wrong request.

- 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):

- ID_PIV = Sender ID of the encrypting endpoint
- 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):

- ID_PIV = Sender ID of the endpoint generating the request
- 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

- 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.

- 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.

- 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.

- 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.

- Length. This field contain 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

- 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.

- 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.

- 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.

- 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).

- 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.

- 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.

- 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:

- 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);

- a choice between two types, by giving both types separated by a "/";

- 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: January 9, 2020

Z. Shelby
ARM
M. Koster
SmartThings
C. Bormann
Universitaet Bremen TZI
P. van der Stok
consultant
C. Amsuess, Ed.
July 08, 2019

CoRE Resource Directory
draft-ietf-core-resource-directory-23

Abstract

In many IoT applications, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which contains information about resources held on other servers, allowing lookups to be performed for those resources. The input to an RD is composed of links and the output is composed of links constructed from the information stored in the RD. This document specifies the web interfaces that a Resource Directory supports for web servers to discover the RD and to register, maintain, lookup and remove information on resources. Furthermore, new target attributes useful in conjunction with an RD are defined.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 9, 2020.

Table of Contents

1. Introduction

In the work on Constrained RESTful Environments (CoRE), a REST architecture suitable for constrained nodes (e.g. with limited RAM and ROM [RFC7228]) and networks (e.g. 6LoWPAN [RFC4944]) has been established and is used in Internet-of-Things (IoT) or machine-to-machine (M2M) applications such as smart energy and building automation.

The discovery of resources offered by a constrained server is very important in machine-to-machine applications where there are no humans in the loop and static interfaces result in fragility. The discovery of resources provided by an HTTP Web Server is typically called Web Linking [RFC8288]. The use of Web Linking for the description and discovery of resources hosted by constrained web servers is specified by the CoRE Link Format [RFC6690]. However, [RFC6690] only describes how to discover resources from the web server that hosts them by querying "/.well-known/core". In many constrained scenarios, direct discovery of resources is not practical due to sleeping nodes, disperse networks, or networks where multicast traffic is inefficient. These problems can be solved by employing an entity called a Resource Directory (RD), which contains information about resources held on other servers, allowing lookups to be performed for those resources.

This document specifies the web interfaces that a Resource Directory supports for web servers to discover the RD and to register, maintain, lookup and remove information on resources. Furthermore, new target attributes useful in conjunction with a Resource Directory are defined. Although the examples in this document show the use of these interfaces with CoAP [RFC7252], they can be applied in an equivalent manner to HTTP [RFC7230].

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119]. The term "byte" is used in its now customary sense as a synonym for "octet".

This specification requires readers to be familiar with all the terms and concepts that are discussed in [RFC3986], [RFC8288] and [RFC6690]. Readers should also be familiar with the terms and concepts discussed in [RFC7252]. To describe the REST interfaces defined in this specification, the URI Template format is used [RFC6570].

This specification makes use of the following additional terminology:

resolve against
: The expression "a URI-reference is _resolved against_ a base URI" is used to describe the process of [RFC3986] Section 5.2. Noteworthy corner cases are that if the URI-reference is a (full) URI and resolved against any base URI, that gives the original full URI, and that resolving an empty URI reference gives the base URI without any fragment identifier.

Resource Directory
: A web entity that stores information about web resources and implements the REST interfaces defined in this specification for registration and lookup of those resources.

Sector
: In the context of a Resource Directory, a sector is a logical grouping of endpoints.

  The abbreviation "d=" is used for the sector in query parameters for compatibility with deployed implementations.

Endpoint
: Endpoint (EP) is a term used to describe a web server or client in [RFC7252]. In the context of this specification an endpoint is used to describe a web server that registers resources to the Resource Directory. An endpoint is identified by its endpoint name, which is included during registration, and has a unique name within the associated sector of the registration.

Registration Base URI
: The Base URI of a Registration is a URI that typically gives scheme and authority information about an Endpoint. The Registration Base URI is provided at registration time, and is used by the Resource Directory to resolve relative references of the registration into URIs.

Target
: The target of a link is the destination address (URI) of the link. It is sometimes identified with "href=", or displayed as "<target>". Relative targets need resolving with respect to the Base URI (section 5.2 of [RFC3986]).

  This use of the term Target is consistent with [RFC8288]’s use of the term.

Context
: The context of a link is the source address (URI) of the link, and describes which resource is linked to the target. A link’s context is made explicit in serialized links as the "anchor=" attribute.

  This use of the term Context is consistent with [RFC8288]’s use of the term.

Directory Resource
: A resource in the Resource Directory (RD) containing registration resources.

Registration Resource
: A resource in the RD that contains information about an Endpoint and its links.

Commissioning Tool
: Commissioning Tool (CT) is a device that assists during the installation of the network by assigning values to parameters, naming endpoints and groups, or adapting the installation to the needs of the applications.

Registrant-ep
: Registrant-ep is the endpoint that is registered into the RD. The registrant-ep can register itself, or a CT registers the registrant-ep.

RDAO
: Resource Directory Address Option. A new IPv6 Neighbor Discovery option defined for announcing a Resource Directory's address.

3. Architecture and Use Cases

3.1. Principles

The Resource Directory is primarily a tool to make discovery operations more efficient than querying /.well-known/core on all connected devices, or across boundaries that would be limiting those operations.

It provides information about resources hosted by other devices that could otherwise only be obtained by directly querying the /.well-known/core resource on these other devices, either by a unicast request or a multicast request.

Information SHOULD only be stored in the resource directory if it can be obtained by querying the described device's /.well-known/core resource directly.

Data in the resource directory can only be provided by the device which hosts those data or a dedicated Commissioning Tool (CT). These CTs are thought to act on behalf of endpoints too constrained, or generally unable, to present that information themselves. No other client can modify data in the resource directory. Changes to the information in the Resource Directory do not propagate automatically back to the web servers from where the information originated.

3.2. Architecture

The resource directory architecture is illustrated in Figure 1. A Resource Directory (RD) is used as a repository of registrations describing resources hosted on other web servers, also called endpoints (EP). An endpoint is a web server associated with a scheme, IP address and port. A physical node may host one or more endpoints. The RD implements a set of REST interfaces for endpoints to register and maintain resource directory registrations, and for endpoints to lookup resources from the RD. An RD can be logically segmented by the use of Sectors.

A mechanism to discover an RD using CoRE Link Format [RFC6690] is defined.

Registrations in the RD are soft state and need to be periodically refreshed.

An endpoint uses specific interfaces to register, update and remove a registration. It is also possible for an RD to fetch Web Links from endpoints and add their contents to resource directory registrations.

At the first registration of an endpoint, a "registration resource" is created, the location of which is returned to the registering endpoint. The registering endpoint uses this registration resource to manage the contents of registrations.

A lookup interface for discovering any of the Web Links stored in the RD is provided using the CoRE Link Format.

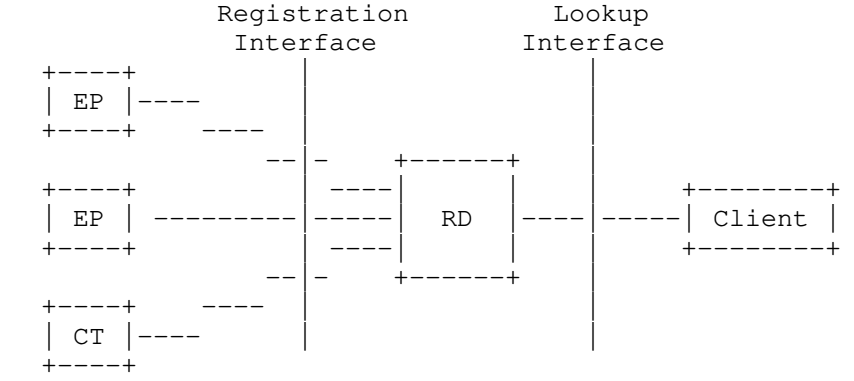


Figure 1: The resource directory architecture.

A Registrant-EP MAY keep concurrent registrations to more than one RD at the same time if explicitly configured to do so, but that is not expected to be supported by typical EP implementations. Any such registrations are independent of each other. The usual expectation when multiple discovery mechanisms or addresses are configured is that they constitute a fall-back path for a single registration.

3.3. RD Content Model

The Entity-Relationship (ER) models shown in Figure 2 and Figure 3 model the contents of /.well-known/core and the resource directory respectively, with entity-relationship diagrams [ER]. Entities (rectangles) are used for concepts that exist independently. Attributes (ovals) are used for concepts that exist only in connection with a related entity. Relations (diamonds) give a semantic meaning to the relation between entities. Numbers specify the cardinality of the relations.

Some of the attribute values are URIs. Those values are always full URIs and never relative references in the information model. They can, however, be expressed as relative references in serializations, and often are.

These models provide an abstract view of the information expressed in link-format documents and a Resource Directory. They cover the concepts, but not necessarily all details of an RD's operation; they are meant to give an overview, and not be a template for implementations.

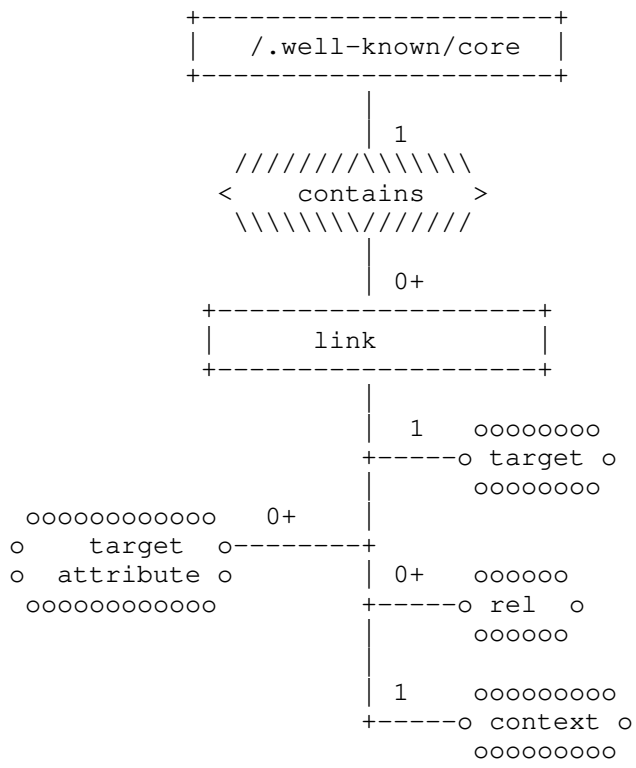


Figure 2: E-R Model of the content of /.well-known/core

The model shown in Figure 2 models the contents of /.well-known/core which contains:

- 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. It defaults to that document's URI.

- 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.

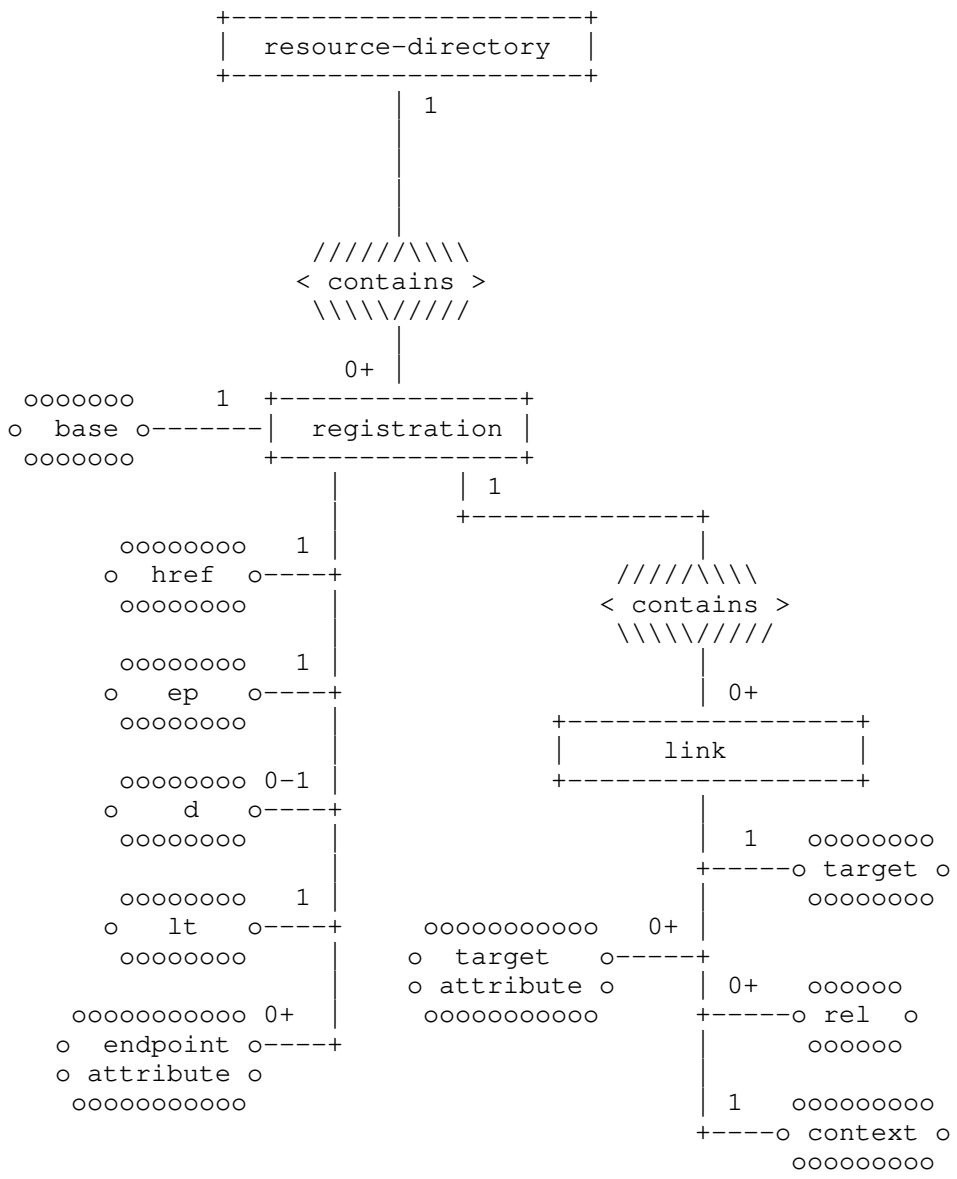


Figure 3: E-R Model of the content of the Resource Directory

The model shown in Figure 3 models the contents of the resource directory which contains in addition to /.well-known/core:

o 0 to n Registrations of endpoints,

A registration is associated with one endpoint. A registration defines a set of links as defined for /.well-known/core. A Registration has six types of attributes:

- 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 can not be serialized to contain zone identifiers (see [RFC6874] Section 1).

Link-local addresses can only be used on a single link (therefore RD servers can not announce them when queried on a different link), and lookup clients using them need to keep track of which interface they got them from.

Therefore, it is advisable in many scenarios to use addresses with larger scope if available.

3.5. Use Case: Cellular M2M

Over the last few years, mobile operators around the world have focused on development of M2M solutions in order to expand the business to the new type of users: machines. The machines are connected directly to a mobile network using an appropriate embedded wireless interface (GSM/GPRS, WCDMA, LTE) or via a gateway providing short and wide range wireless interfaces. From the system design point of view, the ambition is to design horizontal solutions that

can enable utilization of machines in different applications depending on their current availability and capabilities as well as application requirements, thus avoiding silo like solutions. One of the crucial enablers of such design is the ability to discover resources (machines -- endpoints) capable of providing required information at a given time or acting on instructions from the end users.

Imagine a scenario where endpoints installed on vehicles enable tracking of the position of these vehicles for fleet management purposes and allow monitoring of environment parameters. During the boot-up process endpoints register with a Resource Directory, which is hosted by the mobile operator or somewhere in the cloud. Periodically, these endpoints update their registration and may modify resources they offer.

When endpoints are not always connected, for example because they enter a sleep mode, a remote server is usually used to provide proxy access to the endpoints. Mobile apps or web applications for environment monitoring contact the RD, look up the endpoints capable of providing information about the environment using an appropriate set of link parameters, obtain information on how to contact them (URLs of the proxy server), and then initiate interaction to obtain information that is finally processed, displayed on the screen and usually stored in a database. Similarly, fleet management systems provide the appropriate link parameters to the RD to look up for EPs deployed on the vehicles the application is responsible for.

3.6. Use Case: Home and Building Automation

Home and commercial building automation systems can benefit from the use of M2M web services. The discovery requirements of these applications are demanding. Home automation usually relies on run-time discovery to commission the system, whereas in building automation a combination of professional commissioning and run-time discovery is used. Both home and building automation involve peer-to-peer interactions between endpoints, and involve battery-powered sleeping devices.

Two phases can be discerned for a network servicing the system: (1) installation and (2) operation. During the operational phase, the network is connected to the Internet with a Border router (6LBR) and the nodes connected to the network can use the Internet services that are provided by the Internet Provider or the network administrator. During the installation phase, the network is completely stand-alone, no 6LBR is connected, and the network only supports the IP communication between the connected nodes. The installation phase is usually followed by the operational phase.

3.7. Use Case: Link Catalogues

Resources may be shared through data brokers that have no knowledge beforehand of who is going to consume the data. Resource Directory can be used to hold links about resources and services hosted anywhere to make them discoverable by a general class of applications.

For example, environmental and weather sensors that generate data for public consumption may provide data to an intermediary server, or broker. Sensor data are published to the intermediary upon changes or at regular intervals. Descriptions of the sensors that resolve to links to sensor data may be published to a Resource Directory. Applications wishing to consume the data can use RD Lookup to discover and resolve links to the desired resources and endpoints. The Resource Directory service need not be coupled with the data intermediary service. Mapping of Resource Directories to data intermediaries may be many-to-many.

Metadata in web link formats like [RFC6690] which may be internally stored as triples, or relation/attribute pairs providing metadata about resource links, need to be supported by Resource Directories . External catalogues that are represented in other formats may be converted to common web linking formats for storage and access by Resource Directories. Since it is common practice for these to be URN encoded, simple and lossless structural transforms should generally be sufficient to store external metadata in Resource Directories.

The additional features of Resource Directory allow sectors to be defined to enable access to a particular set of resources from particular applications. This provides isolation and protection of sensitive data when needed. Application groups with multicast addresses may be defined to support efficient data transport.

4. RD discovery and other interface-independent components

This and the following sections define the required set of REST interfaces between a Resource Directory (RD), endpoints and lookup clients. Although the examples throughout these sections assume the use of CoAP [RFC7252], these REST interfaces can also be realized using HTTP [RFC7230]. Only multicast discovery operations are not possible on HTTP, and Simple Registration can not be executed as base attribute (which is mandatory for HTTP) can not be used there. In all definitions in these sections, both CoAP response codes (with dot notation) and HTTP response codes (without dot notation) are shown. An RD implementing this specification MUST support the discovery, registration, update, lookup, and removal interfaces.

All operations on the contents of the Resource Directory MUST be atomic and idempotent.

For several operations, interface templates are given in list form; those describe the operation participants, request codes, URIs, content formats and outcomes. Sections of those templates contain normative content about Interaction, Method, URI Template and URI Template Variables as well as the details of the Success condition. The additional sections on options like Content-Format and on Failure codes give typical cases that an implementation of the RD should deal with. Those serve to illustrate the typical responses to readers who are not yet familiar with all the details of CoAP based interfaces; they do not limit what a server may respond under atypical circumstances.

REST clients (registrant-EPs / CTs, lookup clients, RD servers during simple registrations) MUST be prepared to receive any unsuccessful code and act upon it according to its definition, options and/or payload to the best of their capabilities, falling back to failing the operation if recovery is not possible. In particular, they should retry the request upon 5.03 (Service Unavailable; 503 in HTTP) according to the Max-Age (Retry-After in HTTP) option, and fall back to link-format when receiving 4.15 (Unsupported Content Format; 415 in HTTP).

A resource directory MAY make the information submitted to it available to further directories, if it can ensure that a loop does not form. The protocol used between directories to ensure loop-free operation is outside the scope of this document.

4.1. Finding a Resource Directory

A (re-)starting device may want to find one or more resource directories for discovery purposes. Dependent on the operational conditions, one or more of the techniques below apply. The use of DNS-SD [RFC6763] is described in [I-D.ietf-core-rd-dns-sd].

The device may be pre-configured to exercise specific mechanisms for finding the resource directory:

1. It may be configured with a specific IP address for the RD. That IP address may also be an anycast address, allowing the network to forward RD requests to an RD that is topologically close; each target network environment in which some of these preconfigured nodes are to be brought up is then configured with a route for this anycast address that leads to an appropriate RD. (Instead of using an anycast address, a multicast address can also be

preconfigured. The RD servers then need to configure one of their interfaces with this multicast address.)

2. It may be configured with a DNS name for the RD and use DNS to return the IP address of the RD; it can find a DNS server to perform the lookup using the usual mechanisms for finding DNS servers.

For cases where the device is not specifically configured with a way to find a resource directory, the network may want to provide a suitable default.

1. If the address configuration of the network is performed via SLAAC, this is provided by the RDAO option Section 4.1.1.

2. If the address configuration of the network is performed via DHCP, this could be provided via a DHCP option (no such option is defined at the time of writing).

Finally, if neither the device nor the network offers any specific configuration, the device may want to employ heuristics to find a suitable resource directory.

The present specification does not fully define these heuristics, but suggests a number of candidates:

1. In a 6LoWPAN, just assume the Border Router (6LBR) can act as a resource directory (using the ABRO option to find that [RFC6775]). Confirmation can be obtained by sending a Unicast to "coap://[6LBR]/.well-known/core?rt=core.rd*".

2. In a network that supports multicast well, discovering the RD using a multicast query for /.well-known/core as specified in CoRE Link Format [RFC6690]: Sending a Multicast GET to "coap://[MCD1]/.well-known/core?rt=core.rd*". RDs within the multicast scope will answer the query.

When answering a multicast request directed at a link-local address, the RD may want to respond from a routable address; this makes it easier for registrants to use one of their own routable addresses for registration.

As some of the RD addresses obtained by the methods listed here are just (more or less educated) guesses, endpoints MUST make use of any error messages to very strictly rate-limit requests to candidate IP addresses that don’t work out. For example, an ICMP Destination Unreachable message (and, in particular, the port unreachable code for this message) may indicate the lack of a CoAP server on the

candidate host, or a CoAP error response code such as 4.05 "Method Not Allowed" may indicate unwillingness of a CoAP server to act as a directory server.

The following RD discovery mechanisms are recommended:

- 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).

- The use of DNS facilities is described in [I-D.ietf-core-rd-dns-sd].

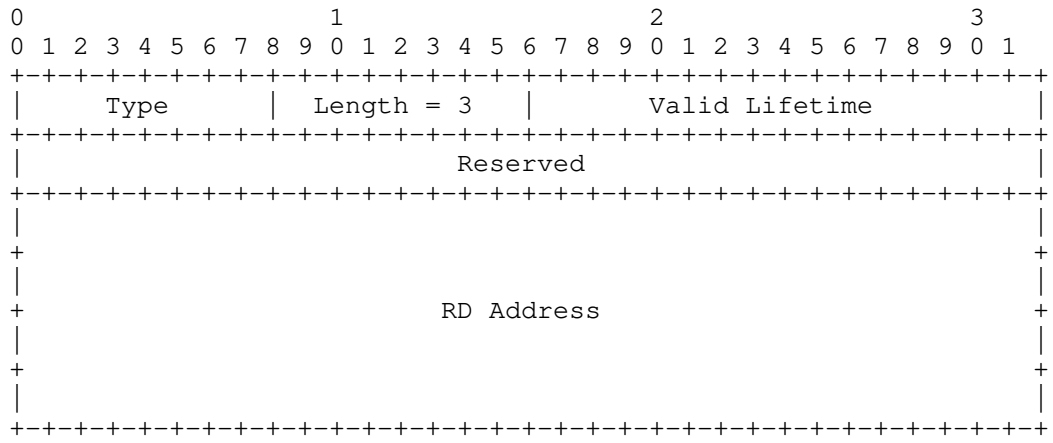
The use of multicast discovery in mesh networks is NOT recommended.

4.1.1. Resource Directory Address Option (RDAO)

The Resource Directory Address Option (RDAO) using IPv6 Neighbor Discovery (ND) carries information about the address of the Resource Directory (RD). This information is needed when endpoints cannot discover the Resource Directory with a link-local or realm-local scope multicast address, for instance because the endpoint and the RD are separated by a Border Router (6LBR). In many circumstances the availability of DHCP cannot be guaranteed either during commissioning of the network. The presence and the use of the RD is essential during commissioning.

It is possible to send multiple RDAO options in one message, indicating as many resource directory addresses.

The RDAO format is:



Fields:

Type: 38

Length: 8-bit unsigned integer. The length of the option in units of 8 bytes. Always 3.

Valid Lifetime: 16-bit unsigned integer. The length of time in units of 60 seconds (relative to the time the packet is received) that this Resource Directory address is valid. A value of all zero bits (0x0) indicates that this Resource Directory address is not valid anymore.

Reserved: This field is unused. It MUST be initialized to zero by the sender and MUST be ignored by the receiver.

RD Address: IPv6 address of the RD.

Figure 4: Resource Directory Address Option

4.2. Payload Content Formats

Resource Directory implementations using this specification MUST support the application/link-format content format (ct=40).

Resource Directories implementing this specification MAY support additional content formats.

Any additional content format supported by a Resource Directory implementing this specification SHOULD be able to express all the information expressible in link-format. It MAY be able to express information that is inexpressible in link-format, but those expressions SHOULD be avoided where possible.

4.3. URI Discovery

Before an endpoint can make use of an RD, it must first know the RD's address and port, and the URI path information for its REST APIs. This section defines discovery of the RD and its URIs using the well-known interface of the CoRE Link Format [RFC6690]. A complete set of RD discovery methods is described in Section 4.1.

Discovery of the RD registration URI path is performed by sending either a multicast or unicast GET request to "/.well-known/core" and including a Resource Type (rt) parameter [RFC6690] with the value "core.rd" in the query string. Likewise, a Resource Type parameter value of "core.rd-lookup*" is used to discover the URIs for RD Lookup operations, core.rd* is used to discover all URI paths for RD operations. Upon success, the response will contain a payload with a link format entry for each RD function discovered, indicating the URI of the RD function returned and the corresponding Resource Type. When performing multicast discovery, the multicast IP address used will depend on the scope required and the multicast capabilities of the network (see Section 9.5).

A Resource Directory MAY provide hints about the content-formats it supports in the links it exposes or registers, using the "ct" target attribute, as shown in the example below. Clients MAY use these hints to select alternate content-formats for interaction with the Resource Directory.

HTTP does not support multicast and consequently only unicast discovery can be supported using HTTP. The well-known entry points SHOULD be provided to enable unicast discovery.

An implementation of this resource directory specification MUST support query filtering for the rt parameter as defined in [RFC6690].

While the link targets in this discovery step are often expressed in path-absolute form, this is not a requirement. Clients of the RD SHOULD therefore accept URIs of all schemes they support, both as URIs and relative references, and not limit the set of discovered URIs to those hosted at the address used for URI discovery.

The URI Discovery operation can yield multiple URIs of a given resource type. The client of the RD can use any of the discovered addresses initially.

The discovery request interface is specified as follows (this is exactly the Well-Known Interface of [RFC6690] Section 4, with the additional requirement that the server MUST support query filtering):

Interaction: EP and Client -> RD

Method: GET

URI Template: /.well-known/core{?rt}

URI Template Variables:

rt := Resource Type. SHOULD contain one of the values "core.rd", "core.rd-lookup*", "core.rd-lookup-res", "core.rd-lookup-ep", or "core.rd*"

Accept: absent, application/link-format or any other media type representing web links

The following response is expected on this interface:

Success: 2.05 "Content" or 200 "OK" with an application/link-format or other web link payload containing one or more matching entries for the RD resource.

The following example shows an endpoint discovering an RD using this interface, thus learning that the directory resource location, in this example, is /rd, and that the content-format delivered by the server hosting the resource is application/link-format (ct=40). Note that it is up to the RD to choose its RD locations.

```
Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content
</rd>;rt="core.rd";ct=40,
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct=40,
</rd-lookup/res>;rt="core.rd-lookup-res";ct=40,
```

Figure 5: Example discovery exchange

The following example shows the way of indicating that a client may request alternate content-formats. The Content-Format code attribute "ct" MAY include a space-separated sequence of Content-Format codes as specified in Section 7.2.1 of [RFC7252], indicating that multiple

content-formats are available. The example below shows the required Content-Format 40 (application/link-format) indicated as well as a CBOR and JSON representation from [I-D.ietf-core-links-json] (which have no numeric values assigned yet, so they are shown as TBD64 and TBD504 as in that draft). The RD resource locations /rd, and /rd-lookup are example values. The server in this example also indicates that it is capable of providing observation on resource lookups.

```
Req: GET coap://[MCD1]/.well-known/core?rt=core.rd*

Res: 2.05 Content
</rd>;rt="core.rd";ct="40 65225",
</rd-lookup/res>;rt="core.rd-lookup-res";ct="40 TBD64 TBD504";obs,
</rd-lookup/ep>;rt="core.rd-lookup-ep";ct="40 TBD64 TBD504",
```

Figure 6: Example discovery exchange indicating additional content-formats

From a management and maintenance perspective, it is necessary to identify the components that constitute the RD server. The identification refers to information about for example client-server incompatibilities, supported features, required updates and other aspects. The URI discovery address, a described in section 4 of [RFC6690] can be used to find the identification.

It would typically be stored in an implementation information link (as described in [I-D.bormann-t2trg-rel-impl]):

```
Req: GET /.well-known/core?rel=impl-info

Res: 2.05 Content
<http://software.example.com/shiny-resource-directory/1.0beta1>;
    rel="impl-info"
```

Figure 7: Example exchange of obtaining implementation information

Note that depending on the particular server's architecture, such a link could be anchored at the RD server's root, at the discovery site (as in this example) or at individual RD components. The latter is to be expected when different applications are run on the same server.

5. Registration

After discovering the location of an RD, a registrant-ep or CT MAY register the resources of the registrant-ep using the registration interface. This interface accepts a POST from an endpoint containing the list of resources to be added to the directory as the message

payload in the CoRE Link Format [RFC6690] or other representations of web links, along with query parameters indicating the name of the endpoint, and optionally the sector, lifetime and base URI of the registration. It is expected that other specifications will define further parameters (see Section 9.3). The RD then creates a new registration resource in the RD and returns its location. The receiving endpoint MUST use that location when refreshing registrations using this interface. Registration resources in the RD are kept active for the period indicated by the lifetime parameter. The creating endpoint is responsible for refreshing the registration resource within this period using either the registration or update interface. The registration interface MUST be implemented to be idempotent, so that registering twice with the same endpoint parameters ep and d (sector) does not create multiple registration resources.

The following rules apply for a registration request targeting a given (ep, d) value pair:

- 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.

The posted link-format document can (and typically does) contain relative references both in its link targets and in its anchors, or contain empty anchors. The RD server needs to resolve these references in order to faithfully represent them in lookups. They are resolved against the base URI of the registration, which is provided either explicitly in the "base" parameter or constructed implicitly from the requester's URI as constructed from its network address and scheme.

For media types to which Appendix C applies (i.e. documents in application/link-format), the RD only needs to accept representations in Limited Link Format as described there. Its behavior with representations outside that subset is implementation defined.

The registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: {+rd}{?ep,d,lt,base,extra-attrs*}

URI Template Variables:

rd := RD registration URI (mandatory). This is the location of the RD, as obtained from discovery.

ep := Endpoint name (mostly mandatory). The endpoint name is an identifier that MUST be unique within a sector. As the endpoint name is a Unicode string, it is encoded in UTF-8 (and possibly pct-encoding) during variable expansion (see [RFC6570] Section 3.2.1). The endpoint name MUST NOT contain any character in the inclusive ranges 0-31 or 127-159. The maximum length of this parameter is 63 UTF-8 encoded bytes. If the RD is configured to recognize the endpoint (e.g. based on its security context), the RD assigns an endpoint name based on a set of configuration parameter values.

d := Sector (optional). The sector to which this endpoint belongs. When this parameter is not present, the RD MAY associate the endpoint with a configured default sector or leave it empty. The sector is encoded like the ep parameter, and is limited to 63 UTF-8 encoded bytes as well. The endpoint name and sector name are not set when one or both are set in an accompanying authorization token.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included in the initial registration, a default value of 90000 (25 hours) SHOULD be assumed.

base := Base URI (optional). This parameter sets the base URI of the registration, under which the relative links in the payload are to be interpreted. The specified URI typically does not have a path component of its own, and MUST be suitable as a base URI to resolve any relative references given in the registration. The parameter is therefore usually of the shape "scheme://authority" for HTTP and CoAP URIs. The URI SHOULD NOT have a query or fragment component as any non-empty relative part in a reference would remove those parts from the resulting URI.

In the absence of this parameter the scheme of the protocol, source address and source port of the registration request are assumed. The Base URI is consecutively constructed by concatenating the used protocol's scheme with the characters "://", the requester's source address as an address literal and

":" followed by its port (if it was not the protocol's default one) in analogy to [RFC7252] Section 6.5.

This parameter is mandatory when the directory is filled by a third party such as an commissioning tool.

If the registrant-ep uses an ephemeral port to register with, it MUST include the base parameter in the registration to provide a valid network path.

A registrant that can not be reached by potential lookup clients at the address it registers from (e.g. because it is behind some form of Network Address Translation (NAT)) MUST provide a reachable base address with its registration.

If the Base URI contains a link-local IP literal, it MUST NOT contain a Zone Identifier, and MUST be local to the link on which the registration request is received.

Endpoints that register with a base that contains a path component can not meaningfully use [RFC6690] Link Format due to its prevalence of the Origin concept in relative reference resolution. Those applications should use different representations of links to which Appendix C is not applicable (e.g. [I-D.hartke-t2trg-coral]).

extra-attrs := Additional registration attributes (optional). The endpoint can pass any parameter registered at Section 9.3 to the directory. If the RD is aware of the parameter's specified semantics, it processes it accordingly. Otherwise, it MUST store the unknown key and its value(s) as an endpoint attribute for further lookup.

Content-Format: application/link-format or any other indicated media type representing web links

The following response is expected on this interface:

Success: 2.01 "Created" or 201 "Created". The Location-Path option or Location header MUST be included in the response. This location MUST be a stable identifier generated by the RD as it is used for all subsequent operations on this registration resource. The registration resource location thus returned is for the purpose of updating the lifetime of the registration and for maintaining the content of the registered links, including updating and deleting links.

A registration with an already registered ep and d value pair responds with the same success code and location as the original registration; the set of links registered with the endpoint is replaced with the links from the payload.

The location MUST NOT have a query or fragment component, as that could conflict with query parameters during the Registration Update operation. Therefore, the Location-Query option MUST NOT be present in a successful response.

If the registration fails, including request timeouts, or if delays from Service Unavailable responses with Max-Age or Retry-After accumulate to exceed the registrant's configured timeouts, it SHOULD pick another registration URI from the "URI Discovery" step and if there is only one or the list is exhausted, pick other choices from the "Finding a Resource Directory" step. Care has to be taken to consider the freshness of results obtained earlier, e.g. of the result of a "/.well-known/core" response, the lifetime of an RDAO option and of DNS responses. Any rate limits and persistent errors from the "Finding a Resource Directory" step must be considered for the whole registration time, not only for a single operation.

The following example shows a registrant-ep with the name "node1" registering two resources to an RD using this interface. The location "/rd" is an example RD location discovered in a request similar to Figure 5.

```
Req: POST coap://rd.example.com/rd?ep=node1
Content-Format: 40
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel="describedby"

Res: 2.01 Created
Location-Path: /rd/4521
```

Figure 8: Example registration payload

A Resource Directory may optionally support HTTP. Here is an example of almost the same registration operation above, when done using HTTP.

```
Req: POST /rd?ep=node1&base=http://[2001:db8:1::1] HTTP/1.1
Host: example.com
Content-Type: application/link-format
Payload:
</sensors/temp>;ct=41;rt="temperature-c";if="sensor",
<http://www.example.com/sensors/temp>;
  anchor="/sensors/temp";rel="describedby"

Res: 201 Created
Location: /rd/4521
```

Figure 9: Example registration payload as expressed using HTTP

5.1. Simple Registration

Not all endpoints hosting resources are expected to know how to upload links to an RD as described in Section 5. Instead, simple endpoints can implement the Simple Registration approach described in this section. An RD implementing this specification MUST implement Simple Registration. However, there may be security reasons why this form of directory discovery would be disabled.

This approach requires that the registrant-ep makes available the hosted resources that it wants to be discovered, as links on its "/.well-known/core" interface as specified in [RFC6690]. The links in that document are subject to the same limitations as the payload of a registration (with respect to Appendix C).

- 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/core" URI of the directory server of choice. The body of the POST request is empty, and triggers the resource directory server to perform GET requests at the requesting registrant-ep's /.well-known/core to obtain the link-format payload to register.

  The registrant-ep includes the same registration parameters in the POST request as it would per Section 5. The registration base URI of the registration is taken from the registrant-ep's network address (as is default with regular registrations).

  Example request from registrant-EP to RD (unanswered until the next step):

```
Req: POST /.well-known/core?lt=6000&ep=node1
(No payload)
```

Figure 10: First half example exchange of a simple registration

- The Resource Directory queries the registrant-ep's discovery resource to determine the success of the operation. It SHOULD keep a cache of the discovery resource and not query it again as long as it is fresh.

Example request from the RD to the registrant-EP:

```
Req: GET /.well-known/core
Accept: 40

Res: 2.05 Content
Content-Format: 40
Payload:
</sen/temp>
```

Figure 11: Example exchange of the RD querying the simple endpoint

With this response, the RD would answer the previous step's request:

```
Res: 2.04 Changed
```

Figure 12: Second half example exchange of a simple registration

The sequence of fetching the registration content before sending a successful response was chosen to make responses reliable, and the caching item was chosen to still allow very constrained registrants. Registrants MUST be able to serve a GET request to "/.well-known/core" after having requested registration. Constrained devices MAY regard the initial request as temporarily failed when they need RAM occupied by their own request to serve the RD's GET, and retry later when the RD already has a cached representation of their discovery resources. Then, the RD can reply immediately and the registrant can receive the response.

The simple registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: /.well-known/core{?ep,d,lt,extra-attrs*}

URI Template Variables are as they are for registration in Section 5. The base attribute is not accepted to keep the registration interface simple; that rules out registration over CoAP-over-TCP or HTTP that would need to specify one.

The following response is expected on this interface:

Success: 2.04 "Changed".

For the second interaction triggered by the above, the registrant-ep takes the role of server and the RD the role of client. (Note that this is exactly the Well-Known Interface of [RFC6690] Section 4):

Interaction: RD -> EP

Method: GET

URI Template: /.well-known/core

The following response is expected on this interface:

Success: 2.05 "Content".

The RD MUST delete registrations created by simple registration after the expiration of their lifetime. Additional operations on the registration resource cannot be executed because no registration location is returned.

5.2. Third-party registration

For some applications, even Simple Registration may be too taxing for some very constrained devices, in particular if the security requirements become too onerous.

In a controlled environment (e.g. building control), the Resource Directory can be filled by a third party device, called a Commissioning Tool (CT). The commissioning tool can fill the Resource Directory from a database or other means. For that purpose scheme, IP address and port of the URI of the registered device is the value of the "base" parameter of the registration described in Section 5.

It should be noted that the value of the "base" parameter applies to all the links of the registration and has consequences for the anchor value of the individual links as exemplified in Appendix B. An eventual (currently non-existing) "base" attribute of the link is not affected by the value of "base" parameter in the registration.

5.3. Operations on the Registration Resource

This section describes how the registering endpoint can maintain the registrations that it created. The registering endpoint can be the registrant-ep or the CT. An endpoint SHOULD NOT use this interface for registrations that it did not create. The registrations are resources of the RD.

After the initial registration, the registering endpoint retains the returned location of the Registration Resource for further operations, including refreshing the registration in order to extend the lifetime and "keep-alive" the registration. When the lifetime of the registration has expired, the RD SHOULD NOT respond to discovery queries concerning this endpoint. The RD SHOULD continue to provide access to the Registration Resource after a registration time-out occurs in order to enable the registering endpoint to eventually refresh the registration. The RD MAY eventually remove the registration resource for the purpose of garbage collection. If the Registration Resource is removed, the corresponding endpoint will need to be re-registered.

The Registration Resource may also be used cancel the registration using DELETE, and to perform further operations beyond the scope of this specification.

These operations are described below.

5.3.1. Registration Update

The update interface is used by the registering endpoint to refresh or update its registration with an RD. To use the interface, the registering endpoint sends a POST request to the registration resource returned by the initial registration operation.

An update MAY update the lifetime or the base URI registration parameters "lt", "base" as in Section 5. Parameters that are not being changed SHOULD NOT be included in an update. Adding parameters that have not changed increases the size of the message but does not have any other implications. Parameters MUST be included as query parameters in an update operation as in Section 5.

A registration update resets the timeout of the registration to the (possibly updated) lifetime of the registration, independent of whether a "lt" parameter was given.

If the base URI of the registration is changed in an update, relative references submitted in the original registration or later updates are resolved anew against the new base.

The registration update operation only describes the use of POST with an empty payload. Future standards might describe the semantics of using content formats and payloads with the POST method to update the links of a registration (see Section 5.3.3).

The update registration request interface is specified as follows:

Interaction: EP -> RD

Method: POST

URI Template: {+location}{?lt,base,extra-attrs*}

URI Template Variables:

location := This is the Location returned by the RD as a result of a successful earlier registration.

lt := Lifetime (optional). Lifetime of the registration in seconds. Range of 60-4294967295. If no lifetime is included, the previous last lifetime set on a previous update or the original registration (falling back to 90000) SHOULD be used.

base := Base URI (optional). This parameter updates the Base URI established in the original registration to a new value. If the parameter is set in an update, it is stored by the RD as the new Base URI under which to interpret the relative links present in the payload of the original registration, following the same restrictions as in the registration. If the parameter is not set in the request but was set before, the previous Base URI value is kept unmodified. If the parameter is not set in the request and was not set before either, the source address and source port of the update request are stored as the Base URI.

extra-attrs := Additional registration attributes (optional). As with the registration, the RD processes them if it knows their semantics. Otherwise, unknown attributes are stored as endpoint attributes, overriding any previously stored endpoint attributes of the same key.

Note that this default behavior does not allow removing an endpoint attribute in an update. For attributes whose functionality depends on the endpoints' ability to remove them in an update, it can make sense to define a value whose presence is equivalent to the absence of a value. As an alternative, an extension can define different updating rules for their attributes. That necessitates either discovery of

whether the RD is aware of that extension, or tolerating the default behavior.

Content-Format: none (no payload)

The following responses are expected on this interface:

Success: 2.04 "Changed" or 204 "No Content" if the update was successfully processed.

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may have been removed).

If the registration fails in any way, including "Not Found" and request timeouts, or if the time indicated in a Service Unabailable Max-Age/Retry-After exceeds the remaining lifetime, the registering endpoint SHOULD attempt registration again.

The following example shows how the registering endpoint updates its registration resource at an RD using this interface with the example location value: /rd/4521.

Req: POST /rd/4521

Res: 2.04 Changed

Figure 13: Example update of a registration

The following example shows the registering endpoint updating its registration resource at an RD using this interface with the example location value: /rd/4521. The initial registration by the registering endpoint set the following values:

- endpoint name (ep)=endpoint1
- lifetime (lt)=500
- Base URI (base)=coap://local-proxy-old.example.com:5683
- payload of Figure 8

The initial state of the Resource Directory is reflected in the following request:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.01 Content
Payload:
<coap://local-proxy-old.example.com:5683/sensors/temp>;ct=41;
    rt="temperature-c";if="sensor";
    anchor="coap://local-proxy-old.example.com:5683/",
<http://www.example.com/sensors/temp>;
    anchor="coap://local-proxy-old.example.com:5683/sensors/temp";rel="described
by"
```

Figure 14: Example lookup before a change to the base address

The following example shows the registering endpoint changing the Base URI to "coaps://new.example.com:5684":

```
Req: POST /rd/4521?base=coaps://new.example.com:5684

Res: 2.04 Changed
```

Figure 15: Example registration update that changes the base address

The consecutive query returns:

```
Req: GET /rd-lookup/res?ep=endpoint1

Res: 2.01 Content
Payload:
<coap://new.example.com:5684/sensors/temp>;ct=41;
    rt="temperature-c";if="sensor";
    anchor="coap://new.example.com:5684/",
<http://www.example.com/sensors/temp>;
    anchor="coap://new.example.com:5684/sensors/temp";rel="describedby"
```

Figure 16: Example lookup after a change to the base address

5.3.2. Registration Removal

Although RD registrations have soft state and will eventually timeout after their lifetime, the registering endpoint SHOULD explicitly remove an entry from the RD if it knows it will no longer be available (for example on shut-down). This is accomplished using a removal interface on the RD by performing a DELETE on the endpoint resource.

The removal request interface is specified as follows:

Interaction: EP -> RD

Method: DELETE

URI Template: {+location}

URI Template Variables:

location := This is the Location returned by the RD as a result of a successful earlier registration.

The following responses are expected on this interface:

Success: 2.02 "Deleted" or 204 "No Content" upon successful deletion

Failure: 4.04 "Not Found" or 404 "Not Found". Registration does not exist (e.g. may already have been removed).

The following examples shows successful removal of the endpoint from the RD with example location value /rd/4521.

Req: DELETE /rd/4521

Res: 2.02 Deleted

Figure 17: Example of a registration removal

5.3.3. Further operations

Additional operations on the registration can be specified in future documents, for example:

- 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.

6. RD Lookup

To discover the resources registered with the RD, a lookup interface must be provided. This lookup interface is defined as a default, and it is assumed that RDs may also support lookups to return resource descriptions in alternative formats (e.g. JSON or CBOR link format [I-D.ietf-core-links-json]) or using more advanced interfaces (e.g. supporting context or semantic based lookup) on different resources that are discovered independently.

RD Lookup allows lookups for endpoints and resources using attributes defined in this document and for use with the CoRE Link Format. The result of a lookup request is the list of links (if any) corresponding to the type of lookup. Thus, an endpoint lookup MUST return a list of endpoints and a resource lookup MUST return a list of links to resources.

The lookup type is selected by a URI endpoint, which is indicated by a Resource Type as per Table 1 below:

Lookup Type	Resource Type	Mandatory
Resource	core.rd-lookup-res	Mandatory
Endpoint	core.rd-lookup-ep	Mandatory

Table 1: Lookup Types

6.1. Resource lookup

Resource lookup results in links that are semantically equivalent to the links submitted to the RD. The links and link parameters returned by the lookup are equal to the submitted ones, except that the target and anchor references are fully resolved.

Links that did not have an anchor attribute are therefore returned with the base URI of the registration as the anchor. Links of which href or anchor was submitted as a (full) URI are returned with these attributes unmodified.

Above rules allow the client to interpret the response as links without any further knowledge of the storage conventions of the RD. The Resource Directory MAY replace the registration base URIs with a configured intermediate proxy, e.g. in the case of an HTTP lookup interface for CoAP endpoints.

If the base URI of a registration contains a link-local address, the RD MUST NOT show its links unless the lookup was made from the same link. The RD MUST NOT include zone identifiers in the resolved URIs.

6.2. Lookup filtering

Using the Accept Option, the requester can control whether the returned list is returned in CoRE Link Format ("application/link-format", default) or in alternate content-formats (e.g. from [I-D.ietf-core-links-json]).

The page and count parameters are used to obtain lookup results in specified increments using pagination, where count specifies how many links to return and page specifies which subset of links organized in sequential pages, each containing 'count' links, starting with link zero and page zero. Thus, specifying count of 10 and page of 0 will return the first 10 links in the result set (links 0-9). Count = 10 and page = 1 will return the next 'page' containing links 10-19, and so on.

Multiple search criteria MAY be included in a lookup. All included criteria MUST match for a link to be returned. The Resource Directory MUST support matching with multiple search criteria.

A link matches a search criterion if it has an attribute of the same name and the same value, allowing for a trailing "*" wildcard operator as in Section 4.1 of [RFC6690]. Attributes that are defined as "link-type" match if the search value matches any of their values (see Section 4.1 of [RFC6690]; e.g. "?if=core.s" matches ";if="abc core.s";"). A resource link also matches a search criterion if its endpoint would match the criterion, and vice versa, an endpoint link matches a search criterion if any of its resource links matches it.

Note that "href" is a valid search criterion and matches target references. Like all search criteria, on a resource lookup it can match the target reference of the resource link itself, but also the registration resource of the endpoint that registered it. Queries for resource link targets MUST be in URI form (i.e. not relative references) and are matched against a resolved link target. Queries for endpoints SHOULD be expressed in path-absolute form if possible and MUST be expressed in URI form otherwise; the RD SHOULD recognize either.

Endpoints that are interested in a lookup result repeatedly or continuously can use mechanisms like ETag caching, resource observation ([RFC7641]), or any future mechanism that might allow more efficient observations of collections. These are advertised, detected and used according to their own specifications and can be used with the lookup interface as with any other resource.

When resource observation is used, every time the set of matching links changes, or the content of a matching link changes, the RD sends a notification with the matching link set. The notification contains the successful current response to the given request, especially with respect to representing zero matching links (see "Success" item below).

The lookup interface is specified as follows:

Interaction: Client -> RD

Method: GET

URI Template: {+type-lookup-location}{?page,count,search*}

URI Template Variables:

type-lookup-location := RD Lookup URI for a given lookup type (mandatory). The address is discovered as described in Section 4.3.

search := Search criteria for limiting the number of results (optional).

page := Page (optional). Parameter cannot be used without the count parameter. Results are returned from result set in pages that contain 'count' links starting from index (page * count). Page numbering starts with zero.

count := Count (optional). Number of results is limited to this parameter value. If the page parameter is also present, the response MUST only include 'count' links starting with the (page * count) link in the result set from the query. If the count parameter is not present, then the response MUST return all matching links in the result set. Link numbering starts with zero.

Accept: absent, application/link-format or any other indicated media type representing web links

The following responses codes are defined for this interface:

Success: 2.05 "Content" or 200 "OK" with an "application/link-format" or other web link payload containing matching entries for the lookup. The payload can contain zero links (which is an empty payload in [RFC6690] link format, but could also be "[]" in JSON based formats), indicating that no entities matched the request.

6.3. Resource lookup examples

The examples in this section assume the existence of CoAP hosts with a default CoAP port 61616. HTTP hosts are possible and do not change the nature of the examples.

The following example shows a client performing a resource lookup with the example resource look-up locations discovered in Figure 5:

```
Req: GET /rd-lookup/res?rt=temperature

Res: 2.05 Content
<coap://[2001:db8:3::123]:61616/temp>;rt="temperature";
           anchor="coap://[2001:db8:3::123]:61616"
```

Figure 18: Example a resource lookup

A client that wants to be notified of new resources as they show up can use observation:

```
Req: GET /rd-lookup/res?rt=light
Observe: 0

Res: 2.05 Content
Observe: 23
Payload: empty

(at a later point in time)

Res: 2.05 Content
Observe: 24
Payload:
<coap://[2001:db8:3::124]/west>;rt="light";
    anchor="coap://[2001:db8:3::124]",
<coap://[2001:db8:3::124]/south>;rt="light";
    anchor="coap://[2001:db8:3::124]",
<coap://[2001:db8:3::124]/east>;rt="light";
    anchor="coap://[2001:db8:3::124]"
```

Figure 19: Example an observing resource lookup

The following example shows a client performing a paginated resource lookup

```
Req: GET /rd-lookup/res?page=0&count=5

Res: 2.05 Content
<coap://[2001:db8:3::123]:61616/res/0>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/1>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/2>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/3>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/4>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616"

Req: GET /rd-lookup/res?page=1&count=5

Res: 2.05 Content
<coap://[2001:db8:3::123]:61616/res/5>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/6>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/7>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/8>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616",
<coap://[2001:db8:3::123]:61616/res/9>;rt=sensor;ct=60;
    anchor="coap://[2001:db8:3::123]:61616"
```

Figure 20: Examples of paginated resource lookup

The following example shows a client performing a lookup of all resources of all endpoints of a given endpoint type. It assumes that two endpoints (with endpoint names "sensor1" and "sensor2") have previously registered with their respective addresses "coap://sensor1.example.com" and "coap://sensor2.example.com", and posted the very payload of the 6th request of section 5 of [RFC6690].

It demonstrates how absolute link targets stay unmodified, while relative ones are resolved:

```
Req: GET /rd-lookup/res?et=oic.d.sensor

<coap://sensor1.example.com/sensors>;ct=40;title="Sensor Index";
    anchor="coap://sensor1.example.com",
<coap://sensor1.example.com/sensors/temp>;rt="temperature-c";
    if="sensor"; anchor="coap://sensor1.example.com",
<coap://sensor1.example.com/sensors/light>;rt="light-lux";
    if="sensor"; anchor="coap://sensor1.example.com",
<http://www.example.com/sensors/t123>;rel="describedby";
    anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor1.example.com/t>;rel="alternate";
    anchor="coap://sensor1.example.com/sensors/temp",
<coap://sensor2.example.com/sensors>;ct=40;title="Sensor Index";
    anchor="coap://sensor2.example.com",
<coap://sensor2.example.com/sensors/temp>;rt="temperature-c";
    if="sensor"; anchor="coap://sensor2.example.com",
<coap://sensor2.example.com/sensors/light>;rt="light-lux";
    if="sensor"; anchor="coap://sensor2.example.com",
<http://www.example.com/sensors/t123>;rel="describedby";
    anchor="coap://sensor2.example.com/sensors/temp",
<coap://sensor2.example.com/t>;rel="alternate";
    anchor="coap://sensor2.example.com/sensors/temp"
```

Figure 21: Example of resource lookup from multiple endpoints

6.4. Endpoint lookup

The endpoint lookup returns registration resources which can only be manipulated by the registering endpoint.

Endpoint registration resources are annotated with their endpoint names (ep), sectors (d, if present) and registration base URI (base; reports the registrant-ep's address if no explicit base was given) as well as a constant resource type (rt="core.rd-ep"); the lifetime (lt) is not reported. Additional endpoint attributes are added as target attributes to their endpoint link unless their specification says otherwise.

Links to endpoints SHOULD be presented in path-absolute form or, if required, as absolute references. (This avoids the RFC6690 ambiguities.)

Base addresses that contain link-local addresses MUST NOT include zone identifiers, and such registrations MUST NOT be shown unless the lookup was made from the same link from which the registration was made.

While Endpoint Lookup does expose the registration resources, the RD does not need to make them accessible to clients. Clients SHOULD NOT attempt to dereference or manipulate them.

A Resource Directory can report endpoints in lookup that are not hosted at the same address. Lookup clients MUST be prepared to see arbitrary URIs as registration resources in the results and treat them as opaque identifiers; the precise semantics of such links are left to future specifications.

The following example shows a client performing an endpoint type (et) lookup with the value oic.d.sensor (which is currently a registered rt value):

Req: GET /rd-lookup/ep?et=oic.d.sensor

Res: 2.05 Content
</rd/1234>;base="coap://[2001:db8:3::127]:61616";ep="node5";
et="oic.d.sensor";ct="40";rt="core.rd-ep",
</rd/4521>;base="coap://[2001:db8:3::129]:61616";ep="node7";
et="oic.d.sensor";ct="40";d="floor-3";rt="core.rd-ep"

Figure 22: Examples of endpoint lookup

7. Security policies

The Resource Directory (RD) provides assistance to applications situated on a selection of nodes to discover endpoints on connected nodes. This section discusses different security aspects of accessing the RD.

The contents of the RD are inserted in two ways:

1. The node hosting the discoverable endpoint fills the RD with the contents of /.well-known/core by:

   * Storing the contents directly into RD (see Section 5)

   * Requesting the RD to load the contents from /.well-known/core (see Section 5.1)

2. A Commissioning Tool (CT) fills the RD with endpoint information for a set of discoverable nodes. (see Section 5 with base=authority parameter value)

In both cases, the nodes filling the RD should be authenticated and authorized to change the contents of the RD. An Authorization Server (AS) is responsible to assign a token to the registering node to

authorize the node to discover or register endpoints in a given RD [I-D.ietf-ace-oauth-authz].

It can be imagined that an installation is divided in a set of security regions, each one with its own RD(s) to discover the endpoints that are part of a given security region. An endpoint that wants to discover an RD, responsible for a given region, needs to be authorized to learn the contents of a given RD. Within a region, for a given RD, a more fine-grained security division is possible based on the values of the endpoint registration parameters. Authorization to discover endpoints with a given set of filter values is recommended for those cases.

When a node registers its endpoints, criteria are needed to authorize the node to enter them. An important aspect is the uniqueness of the (endpoint name, and optional sector) pair within the RD. Consider the two cases separately: (1) CT registers endpoints, and (2) the registering node registers its own endpoint(s).

- A CT needs authorization to register a set of endpoints. This authorization can be based on the region, i.e. a given CT is authorized to register any endpoint (endpoint name, sector) into a given RD, or to register an endpoint with (endpoint name, sector) value pairs assigned by the AS, or can be more fine-grained, including a subset of registration parameter values.

- A given endpoint that registers itself, needs to proof its possession of its unique (endpoint name, sector) value pair. Alternatively, the AS can authorize the endpoint to register with an (endpoint name, sector) value pair assigned by the AS.

A separate document needs to specify these aspects to ensure interoperability between registering nodes and RD. The subsections below give some hints how to handle a subset of the different aspects.

7.1. Secure RD discovery

The Resource Server (RS) discussed in [I-D.ietf-ace-oauth-authz] is equated to the RD. The client (C) needs to discover the RD as discussed in Section 4.1. C can discover the related AS by sending a request to the RD. The RD denies the request by sending the address of the related AS, as discussed in section 5.1 of [I-D.ietf-ace-oauth-authz]. The client MUST send an authorization request to the AS. When appropriate, the AS returns a token that specifies the authorization permission which needs to be specified in a separate document.

7.2. Secure RD filtering

The authorized parameter values for the queries by a given endpoint must be registered by the AS. The AS communicates the parameter values in the token. A separate document needs to specify the parameter value combinations and their storage in the token. The RD decodes the token and checks the validity of the queries of the client.

7.3. Secure endpoint Name assignment

This section only considers the assignment of a name to the endpoint based on an automatic mechanism without use of AS. More elaborate protocols are out of scope. The registering endpoint is authorized by the AS to discover the RD and add registrations. A token is provided by the AS and communicated from registering endpoint to RD. It is assumed that DTLS is used to secure the channel between registering endpoint and RD, where the registering endpoint is the DTLS client. Assuming that the client is provided by a certificate at manufacturing time, the certificate is uniquely identified by the CN field and the serial number. The RD can assign a unique endpoint name by using the certificate identifier as endpoint name. Proof of possession of the endpoint name by the registering endpoint is checked by encrypting the certificate identifier with the private key of the registering endpoint, which the RD can decrypt with the public key stored in the certificate. Even simpler, the authorized registering endpoint can generate a random number (or string) that identifies the endpoint. The RD can check for the improbable replication of the random value. The RD MUST check that registering endpoint uses only one random value for each authorized endpoint.

8. Security Considerations

The security considerations as described in Section 5 of [RFC8288] and Section 6 of [RFC6690] apply. The "/.well-known/core" resource may be protected e.g. using DTLS when hosted on a CoAP server as described in [RFC7252]. DTLS or TLS based security SHOULD be used on all resource directory interfaces defined in this document.

8.1. Endpoint Identification and Authentication

An Endpoint (name, sector) pair is unique within the et of endpoints registered by the RD. An Endpoint MUST NOT be identified by its protocol, port or IP address as these may change over the lifetime of an Endpoint.

Every operation performed by an Endpoint on a resource directory SHOULD be mutually authenticated using Pre-Shared Key, Raw Public Key or Certificate based security.

Consider the following threat: two devices A and B are registered at a single server. Both devices have unique, per-device credentials for use with DTLS to make sure that only parties with authorization to access A or B can do so.

Now, imagine that a malicious device A wants to sabotage the device B. It uses its credentials during the DTLS exchange. Then, it specifies the endpoint name of device B as the name of its own endpoint in device A. If the server does not check whether the identifier provided in the DTLS handshake matches the identifier used at the CoAP layer then it may be inclined to use the endpoint name for looking up what information to provision to the malicious device.

Section 7.3 specifies an example that removes this threat for endpoints that have a certificate installed.

8.2. Access Control

Access control SHOULD be performed separately for the RD registration and Lookup API paths, as different endpoints may be authorized to register with an RD from those authorized to lookup endpoints from the RD. Such access control SHOULD be performed in as fine-grained a level as possible. For example access control for lookups could be performed either at the sector, endpoint or resource level.

8.3. Denial of Service Attacks

Services that run over UDP unprotected are vulnerable to unknowingly become part of a DDoS attack as UDP does not require return routability check. Therefore, an attacker can easily spoof the source IP of the target entity and send requests to such a service which would then respond to the target entity. This can be used for large-scale DDoS attacks on the target. Especially, if the service returns a response that is order of magnitudes larger than the request, the situation becomes even worse as now the attack can be amplified. DNS servers have been widely used for DDoS amplification attacks. There is also a danger that NTP Servers could become implicated in denial-of-service (DoS) attacks since they run on unprotected UDP, there is no return routability check, and they can have a large amplification factor. The responses from the NTP server were found to be 19 times larger than the request. A Resource Directory (RD) which responds to wild-card lookups is potentially vulnerable if run with CoAP over UDP. Since there is no return routability check and the responses can be significantly larger than

requests, RDs can unknowingly become part of a DDoS amplification attack.

9. IANA Considerations

9.1. Resource Types

IANA is asked to enter the following values into the Resource Type (rt=) Link Target Attribute Values sub-registry of the Constrained Restful Environments (CoRE) Parameters registry defined in [RFC6690]:

Value	Description	Reference
core.rd	Directory resource of an RD	RFCTHIS Section 4.3
core.rd-lookup-res	Resource lookup of an RD	RFCTHIS Section 4.3
core.rd-lookup-ep	Endpoint lookup of an RD	RFCTHIS Section 4.3
core.rd-ep	Endpoint resource of an RD	RFCTHIS Section 6

9.2. IPv6 ND Resource Directory Address Option

This document registers one new ND option type under the sub-registry "IPv6 Neighbor Discovery Option Formats":

- Resource Directory Address Option (38)

9.3. RD Parameter Registry

This specification defines a new sub-registry for registration and lookup parameters called "RD Parameters" under "CoRE Parameters". Although this specification defines a basic set of parameters, it is expected that other standards that make use of this interface will define new ones.

Each entry in the registry must include

- 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,

o validity requirements if any, and

o a description.

The query parameter MUST be both a valid URI query key [RFC3986] and a token as used in [RFC8288].

The description must give details on whether the parameter can be updated, and how it is to be processed in lookups.

The mechanisms around new RD parameters should be designed in such a way that they tolerate RD implementations that are unaware of the parameter and expose any parameter passed at registration or updates on in endpoint lookups. (For example, if a parameter used at registration were to be confidential, the registering endpoint should be instructed to only set that parameter if the RD advertises support for keeping it confidential at the discovery step.)

Initial entries in this sub-registry are as follows:

Full name	Short	Validity	Use	Description
Endpoint Name	ep	Unicode*	RLA	Name of the endpoint
Lifetime	lt	60-4294967295	R	Lifetime of the registration in seconds
Sector	d	Unicode*	RLA	Sector to which this endpoint belongs
Registration Base URI	base	URI	RLA	The scheme, address and port and path at which this server is available
Page	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 2: RD Parameters

(Short: Short name used in query parameters or target attributes. Validity: Unicode* = 63 Bytes of UTF-8 encoded Unicode, with no control characters as per Section 5. Use: R = used at registration, L = used at lookup, A = expressed in target attribute

The descriptions for the options defined in this document are only summarized here. To which registrations they apply and when they are to be shown is described in the respective sections of this document.

The IANA policy for future additions to the sub-registry is "Expert Review" as described in [RFC8126]. The evaluation should consider formal criteria, duplication of functionality (Is the new entry redundant with an existing one?), topical suitability (E.g. is the described property actually a property of the endpoint and not a property of a particular resource, in which case it should go into the payload of the registration and need not be registered?), and the potential for conflict with commonly used target attributes (For example, "if" could be used as a parameter for conditional registration if it were not to be used in lookup or attributes, but would make a bad parameter for lookup, because a resource lookup with an "if" query parameter could ambiguously filter by the registered endpoint property or the [RFC6690] target attribute). It is expected that the registry will receive between 5 and 50 registrations in total over the next years.

9.3.1. Full description of the "Endpoint Type" Registration Parameter

An endpoint registering at an RD can describe itself with endpoint types, similar to how resources are described with Resource Types in [RFC6690]. An endpoint type is expressed as a string, which can be either a URI or one of the values defined in the Endpoint Type sub-registry. Endpoint types can be passed in the "et" query parameter as part of extra-attrs at the Registration step, are shown on endpoint lookups using the "et" target attribute, and can be filtered for using "et" as a search criterion in resource and endpoint lookup. Multiple endpoint types are given as separate query parameters or link attributes.

Note that Endpoint Type differs from Resource Type in that it uses multiple attributes rather than space separated values. As a result, Resource Directory implementations automatically support correct filtering in the lookup interfaces from the rules for unknown endpoint attributes.

9.4. "Endpoint Type" (et=) RD Parameter values

This specification establishes a new sub-registry under "CoRE Parameters" called '"Endpoint Type" (et=) RD Parameter values'. The registry properties (required policy, requirements, template) are identical to those of the Resource Type parameters in [RFC6690], in short:

The review policy is IETF Review for values starting with "core", and Specification Required for others.

The requirements to be enforced are:

- 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, RFC 5771).

IPv6 - "all CoRE resource directories" address MCD1 (suggestions FF0X::FE), from the "IPv6 Multicast Address Space Registry", in the "Variable Scope Multicast Addresses" space (RFC 3307). Note that there is a distinct multicast address for each scope that interested CoAP nodes should listen to; CoAP needs the Link-Local and Site-Local scopes only.

[The RFC editor is asked to replace MCD1 and MCD2 with the assigned addresses throughout the document.]

10. Examples

Two examples are presented: a Lighting Installation example in Section 10.1 and a LWM2M example in Section 10.2.

10.1. Lighting Installation

This example shows a simplified lighting installation which makes use of the Resource Directory (RD) with a CoAP interface to facilitate the installation and start-up of the application code in the lights and sensors. In particular, the example leads to the definition of a group and the enabling of the corresponding multicast address as described in Appendix A. No conclusions must be drawn on the realization of actual installation or naming procedures, because the example only "emphasizes" some of the issues that may influence the use of the RD and does not pretend to be normative.

10.1.1. Installation Characteristics

The example assumes that the installation is managed. That means that a Commissioning Tool (CT) is used to authorize the addition of nodes, name them, and name their services. The CT can be connected to the installation in many ways: the CT can be part of the installation network, connected by WiFi to the installation network, or connected via GPRS link, or other method.

It is assumed that there are two naming authorities for the installation: (1) the network manager that is responsible for the correct operation of the network and the connected interfaces, and (2) the lighting manager that is responsible for the correct functioning of networked lights and sensors. The result is the existence of two naming schemes coming from the two managing entities.

The example installation consists of one presence sensor, and two luminaries, luminary1 and luminary2, each with their own wireless interface. Each luminary contains three lamps: left, right and middle. Each luminary is accessible through one endpoint. For each lamp a resource exists to modify the settings of a lamp in a luminary. The purpose of the installation is that the presence sensor notifies the presence of persons to a group of lamps. The group of lamps consists of: middle and left lamps of luminary1 and right lamp of luminary2.

Before commissioning by the lighting manager, the network is installed and access to the interfaces is proven to work by the network manager.

At the moment of installation, the network under installation is not necessarily connected to the DNS infra structure. Therefore, SLAAC IPv6 addresses are assigned to CT, RD, luminaries and sensor shown in Table 3 below:

Name	IPv6 address
luminary1	2001:db8:4::1
luminary2	2001:db8:4::2
Presence sensor	2001:db8:4::3
Resource directory	2001:db8:4::ff

Table 3: interface SLAAC addresses

In Section 10.1.2 the use of resource directory during installation is presented.

10.1.2. RD entries

It is assumed that access to the DNS infrastructure is not always possible during installation. Therefore, the SLAAC addresses are used in this section.

For discovery, the resource types (rt) of the devices are important. The lamps in the luminaries have rt: light, and the presence sensor has rt: p-sensor. The endpoints have names which are relevant to the light installation manager. In this case luminary1, luminary2, and the presence sensor are located in room 2-4-015, where luminary1 is located at the window and luminary2 and the presence sensor are located at the door. The endpoint names reflect this physical location. The middle, left and right lamps are accessed via path /light/middle, /light/left, and /light/right respectively. The identifiers relevant to the Resource Directory are shown in Table 4 below:

Name	endpoint	resource path	resource type
luminary1	lm_R2-4-015_wndw	/light/left	light
luminary1	lm_R2-4-015_wndw	/light/middle	light
luminary1	lm_R2-4-015_wndw	/light/right	light
luminary2	lm_R2-4-015_door	/light/left	light
luminary2	lm_R2-4-015_door	/light/middle	light
luminary2	lm_R2-4-015_door	/light/right	light
Presence sensor	ps_R2-4-015_door	/ps	p-sensor

Table 4: Resource Directory identifiers

It is assumed that the CT knows the RD's address, and has performed URI discovery on it that returned a response like the one in the Section 4.3 example.

The CT inserts the endpoints of the luminaries and the sensor in the RD using the registration base URI parameter (base) to specify the interface address:

```
Req: POST coap://[2001:db8:4::ff]/rd
  ?ep=lm_R2-4-015_wndw&base=coap://[2001:db8:4::1]&d=R2-4-015
Payload:
</light/left>;rt="light",
</light/middle>;rt="light",
</light/right>;rt="light"

Res: 2.01 Created
Location-Path: /rd/4521

Req: POST coap://[2001:db8:4::ff]/rd
  ?ep=lm_R2-4-015_door&base=coap://[2001:db8:4::2]&d=R2-4-015
Payload:
</light/left>;rt="light",
</light/middle>;rt="light",
</light/right>;rt="light"

Res: 2.01 Created
Location-Path: /rd/4522

Req: POST coap://[2001:db8:4::ff]/rd
  ?ep=ps_R2-4-015_door&base=coap://[2001:db8:4::3]d&d=R2-4-015
Payload:
</ps>;rt="p-sensor"

Res: 2.01 Created
Location-Path: /rd/4523
```

Figure 23: Example of registrations a CT enters into an RD

The sector name d=R2-4-015 has been added for an efficient lookup because filtering on "ep" name is more awkward. The same sector name is communicated to the two luminaries and the presence sensor by the CT.

The group is specified in the RD. The base parameter is set to the site-local multicast address allocated to the group. In the POST in the example below, the resources supported by all group members are published.

```
Req: POST coap://[2001:db8:4::ff]/rd
?ep=grp_R2-4-015&et=core.rd-group&base=coap://[ff05::1]
Payload:
</light/left>;rt="light",
</light/middle>;rt="light",
</light/right>;rt="light"

Res: 2.01 Created
Location-Path: /rd/501
```

Figure 24: Example of a multicast group a CT enters into an RD

After the filling of the RD by the CT, the application in the luminaries can learn to which groups they belong, and enable their interface for the multicast address.

The luminary, knowing its sector and being configured to join any group containing lights, searches for candidate groups and joins them:

```
Req: GET coap://[2001:db8:4::ff]/rd-lookup/ep
  ?d=R2-4-015&et=core.rd-group&rt=light

Res: 2.05 Content
</rd/501>;ep="grp_R2-4-015";et="core.rd-group";
          base="coap://[ff05::1]";rt="core.rd-ep"
```

Figure 25: Example of a lookup exchange to find suitable multicast addresses

From the returned base parameter value, the luminary learns the multicast address of the multicast group.

Alternatively, the CT can communicate the multicast address directly to the luminaries by using the "coap-group" resource specified in [RFC7390].

```
Req: POST coap://[2001:db8:4::1]/coap-group
Content-Format: application/coap-group+json
Payload:
{ "a": "[ff05::1]", "n": "grp_R2-4-015"}

Res: 2.01 Created
Location-Path: /coap-group/1
```

Figure 26: Example use of direct multicast address configuration

Dependent on the situation, only the address, "a", or the name, "n", is specified in the coap-group resource.

The presence sensor can learn the presence of groups that support resources with rt=light in its own sector by sending the same request, as used by the luminary. The presence sensor learns the multicast address to use for sending messages to the luminaries.

10.2. OMA Lightweight M2M (LWM2M) Example

This example shows how the OMA LWM2M specification makes use of Resource Directory (RD).

OMA LWM2M is a profile for device services based on CoAP(OMA Name Authority). LWM2M defines a simple object model and a number of abstract interfaces and operations for device management and device service enablement.

An LWM2M server is an instance of an LWM2M middleware service layer, containing a Resource Directory along with other LWM2M interfaces defined by the LWM2M specification.

CoRE Resource Directory (RD) is used to provide the LWM2M Registration interface.

LWM2M does not provide for registration sectors and does not currently use the rd-lookup interface.

The LWM2M specification describes a set of interfaces and a resource model used between a LWM2M device and an LWM2M server. Other interfaces, proxies, and applications are currently out of scope for LWM2M.

The location of the LWM2M Server and RD URI path is provided by the LWM2M Bootstrap process, so no dynamic discovery of the RD is used. LWM2M Servers and endpoints are not required to implement the /.well-known/core resource.

10.2.1. The LWM2M Object Model

The OMA LWM2M object model is based on a simple 2 level class hierarchy consisting of Objects and Resources.

An LWM2M Resource is a REST endpoint, allowed to be a single value or an array of values of the same data type.

An LWM2M Object is a resource template and container type that encapsulates a set of related resources. An LWM2M Object represents

a specific type of information source; for example, there is a LWM2M Device Management object that represents a network connection, containing resources that represent individual properties like radio signal strength.

Since there may potentially be more than one of a given type object, for example more than one network connection, LWM2M defines instances of objects that contain the resources that represent a specific physical thing.

The URI template for LWM2M consists of a base URI followed by Object, Instance, and Resource IDs:

{/base-uri}{/object-id}{/object-instance}{/resource-id}{/resource-instance}

The five variables given here are strings. base-uri can also have the special value "undefined" (sometimes called "null" in RFC 6570). Each of the variables object-instance, resource-id, and resource-instance can be the special value "undefined" only if the values behind it in this sequence also are "undefined". As a special case, object-instance can be "empty" (which is different from "undefined") if resource-id is not "undefined".

base-uri := Base URI for LWM2M resources or "undefined" for default (empty) base URI

object-id := OMNA (OMA Name Authority) registered object ID (0-65535)

object-instance := Object instance identifier (0-65535) or "undefined"/"empty" (see above)) to refer to all instances of an object ID

resource-id := OMNA (OMA Name Authority) registered resource ID (0-65535) or "undefined" to refer to all resources within an instance

resource-instance := Resource instance identifier or "undefined" to refer to single instance of a resource

LWM2M IDs are 16 bit unsigned integers represented in decimal (no leading zeroes except for the value 0) by URI format strings. For example, a LWM2M URI might be:

/1/0/1

The base uri is empty, the Object ID is 1, the instance ID is 0, the resource ID is 1, and the resource instance is "undefined". This example URI points to internal resource 1, which represents the

registration lifetime configured, in instance 0 of a type 1 object (LWM2M Server Object).

10.2.2. LWM2M Register Endpoint

LWM2M defines a registration interface based on the REST API, described in Section 5. The RD registration URI path of the LWM2M Resource Directory is specified to be "/rd".

LWM2M endpoints register object IDs, for example </1>, to indicate that a particular object type is supported, and register object instances, for example </1/0>, to indicate that a particular instance of that object type exists.

Resources within the LWM2M object instance are not registered with the RD, but may be discovered by reading the resource links from the object instance using GET with a CoAP Content-Format of application/link-format. Resources may also be read as a structured object by performing a GET to the object instance with a Content-Format of senml+json.

When an LWM2M object or instance is registered, this indicates to the LWM2M server that the object and its resources are available for management and service enablement (REST API) operations.

LWM2M endpoints may use the following RD registration parameters as defined in Table 2 :

ep - Endpoint Name
lt - registration lifetime

Endpoint Name, Lifetime, and LWM2M Version are mandatory parameters for the register operation, all other registration parameters are optional.

Additional optional LWM2M registration parameters are defined:

Name	Query	Validity	Description
Binding Mode	b	{"U",UQ","S","SQ","US","UQS"}	Available Protocols
LWM2M Version	ver	1.0	Spec Version
SMS Number	sms		MSISDN

Table 5: LWM2M Additional Registration Parameters

The following RD registration parameters are not currently specified for use in LWM2M:

et - Endpoint Type
base - Registration Base URI

The endpoint registration must include a payload containing links to all supported objects and existing object instances, optionally including the appropriate link-format relations.

Here is an example LWM2M registration payload:

</1>,</1/0>,</3/0>,</5>

This link format payload indicates that object ID 1 (LWM2M Server Object) is supported, with a single instance 0 existing, object ID 3 (LWM2M Device object) is supported, with a single instance 0 existing, and object 5 (LWM2M Firmware Object) is supported, with no existing instances.

10.2.3. LWM2M Update Endpoint Registration

The LwM2M update is really very similar to the registration update as described in Section 5.3.1, with the only difference that there are more parameters defined and available. All the parameters listed in that section are also available with the initial registration but are all optional:

lt - Registration Lifetime
b - Protocol Binding
sms - MSISDN
link payload - new or modified links

A Registration update is also specified to be used to update the LWM2M server whenever the endpoint's UDP port or IP address are changed.

10.2.4. LWM2M De-Register Endpoint

LWM2M allows for de-registration using the delete method on the returned location from the initial registration operation. LWM2M de-registration proceeds as described in Section 5.3.2.

11. Acknowledgments

Oscar Novo, Srdjan Krco, Szymon Sasin, Kerry Lynn, Esko Dijk, Anders Brandt, Matthieu Vial, Jim Schaad, Mohit Sethi, Hauke Petersen, Hannes Tschofenig, Sampo Ukkola, Linyi Tian, Jan Newmarch, Matthias Kovatsch, Jaime Jimenez and Ted Lemon have provided helpful comments, discussions and ideas to improve and shape this document. Zach would also like to thank his colleagues from the EU FP7 SENSEI project, where many of the resource directory concepts were originally developed.

12. Changelog

changes from -22 to -23

- 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

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, DOI 10.17487/RFC6570, March 2012, <https://www.rfc-editor.org/info/rfc6570>.

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

[RFC6763] Cheshire, S. and M. Krochmal, "DNS-Based Service Discovery", RFC 6763, DOI 10.17487/RFC6763, February 2013, <https://www.rfc-editor.org/info/rfc6763>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

13.2. Informative References

[ER] Chen, P., "The entity-relationship model---toward a unified view of data", ACM Transactions on Database Systems Vol. 1, pp. 9-36, DOI 10.1145/320434.320440, March 1976.

[I-D.bormann-t2trg-rel-impl] Bormann, C., "impl-info: A link relation type for disclosing implementation information", draft-bormann-t2trg-rel-impl-00 (work in progress), January 2018.

[I-D.hartke-t2trg-coral] Hartke, K., "The Constrained RESTful Application Language (CoRAL)", draft-hartke-t2trg-coral-08 (work in progress), March 2019.

[I-D.ietf-ace-oauth-authz]
Seitz, L., Selander, G., Wahlstroem, E., Erdtman, S., and H. Tschofenig, "Authentication and Authorization for Constrained Environments (ACE) using the OAuth 2.0 Framework (ACE-OAuth)", draft-ietf-ace-oauth-authz-24 (work in progress), March 2019.

[I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing Constrained RESTful Environments (CoRE) Link Format in JSON and CBOR", draft-ietf-core-links-json-10 (work in progress), February 2018.

[I-D.ietf-core-rd-dns-sd]
Stok, P., Koster, M., and C. Amsuess, "CoRE Resource Directory: DNS-SD mapping", draft-ietf-core-rd-dns-sd-05 (work in progress), July 2019.

[I-D.silverajan-core-coap-protocol-negotiation]
Silverajan, B. and M. Ocak, "CoAP Protocol Negotiation", draft-silverajan-core-coap-protocol-negotiation-09 (work in progress), July 2018.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[RFC6775] Shelby, Z., Ed., Chakrabarti, S., Nordmark, E., and C. Bormann, "Neighbor Discovery Optimization for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 6775, DOI 10.17487/RFC6775, November 2012, <https://www.rfc-editor.org/info/rfc6775>.

[RFC6874] Carpenter, B., Cheshire, S., and R. Hinden, "Representing IPv6 Zone Identifiers in Address Literals and Uniform Resource Identifiers", RFC 6874, DOI 10.17487/RFC6874, February 2013, <https://www.rfc-editor.org/info/rfc6874>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <https://www.rfc-editor.org/info/rfc7390>.

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

[RFC8132] van der Stok, P., Bormann, C., and A. Sehgal, "PATCH and FETCH Methods for the Constrained Application Protocol (CoAP)", RFC 8132, DOI 10.17487/RFC8132, April 2017, <https://www.rfc-editor.org/info/rfc8132>.

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

Appendix A. Groups Registration and Lookup

The RD-Groups usage pattern allows announcing application groups inside a Resource Directory.

Groups are represented by endpoint registrations. Their base address is a multicast address, and they SHOULD be entered with the endpoint type "core.rd-group". The endpoint name can also be referred to as a group name in this context.

The registration is inserted into the RD by a Commissioning Tool, which might also be known as a group manager here. It performs third party registration and registration updates.

The links it registers SHOULD be available on all members that join the group. Depending on the application, members that lack some resource MAY be permissible if requests to them fail gracefully.

The following example shows a CT registering a group with the name "lights" which provides two resources. The directory resource path /rd is an example RD location discovered in a request similar to Figure 5.

```
Req: POST coap://rd.example.com/rd?ep=lights&et=core.rd-group
                                    &base=coap://[ff35:30:2001:db8::1]
Content-Format: 40
Payload:
</light>;rt="light";if="core.a",
</color-temperature>;if="core.p";u="K"

Res: 2.01 Created
Location-Path: /rd/12
```

Figure 27: Example registration of a group

In this example, the group manager can easily permit devices that have no writable color-temperature to join, as they would still respond to brightness changing commands. Had the group instead contained a single resource that sets brightness and color temperature atomically, endpoints would need to support both properties.

The resources of a group can be looked up like any other resource, and the group registrations (along with any additional registration parameters) can be looked up using the endpoint lookup interface.

The following example shows a client performing and endpoint lookup for all groups.

```
Req: GET /rd-lookup/ep?et=core.rd-group

Res: 2.01 Content
Payload:
</rd/501>;ep="GRP_R2-4-015";et="core.rd-group";
                                     base="coap://[ff05::1]",
</rd/12>;ep=lights&et=core.rd-group;
         base="coap://[ff35:30:2001:db8::1]";rt="core.rd-ep"
```

Figure 28: Example lookup of groups

The following example shows a client performing a lookup of all resources of all endpoints (groups) with et=core.rd-group.

```
Req: GET /rd-lookup/res?et=core.rd-group

<coap://[ff35:30:2001:db8::1]/light>;rt="light";if="core.a";
     et="core.rd-group";anchor="coap://[ff35:30:2001:db8::1]",
<coap://[ff35:30:2001:db8::1]/color-temperature>;if="core.p";u="K";
     et="core.rd-group";
     anchor="coap://[ff35:30:2001:db8::1]"
```

Figure 29: Example lookup of resources inside groups

Appendix B. Web links and the Resource Directory

Understanding the semantics of a link-format document and its URI references is a journey through different documents ([RFC3986] defining URIs, [RFC6690] defining link-format documents based on [RFC8288] which defines link headers, and [RFC7252] providing the transport). This appendix summarizes the mechanisms and semantics at play from an entry in ".well-known/core" to a resource lookup.

This text is primarily aimed at people entering the field of Constrained Restful Environments from applications that previously did not use web mechanisms.

The explanation of the steps makes some shortcuts in the more confusing details of [RFC6690], which are justified as all examples being in Limited Link Format.

B.1. A simple example

Let's start this example with a very simple host, "2001:db8:f0::1". A client that follows classical CoAP Discovery ([RFC7252] Section 7), sends the following multicast request to learn about neighbours supporting resources with resource-type "temperature".

The client sends a link-local multicast:

```
GET coap://[ff02::fd]:5683/.well-known/core?rt=temperature

RES 2.05 Content
</temp>;rt=temperature;ct=0
```

Figure 30: Example of direct resource discovery

where the response is sent by the server, "[2001:db8:f0::1]:5683".

While the client - on the practical or implementation side - can just go ahead and create a new request to "[2001:db8:f0::1]:5683" with

Uri-Path: "temp", the full resolution steps for insertion into and retrieval from the RD without any shortcuts are:

B.1.1. Resolving the URIs

The client parses the single returned record. The link's target (sometimes called "href") is ""/temp"", which is a relative URI that needs resolving. The base URI <coap://[ff02::fd]:5683/.well-known/core> is used to resolve the reference /temp against.

The Base URI of the requested resource can be composed from the header options of the CoAP GET request by following the steps of [RFC7252] section 6.5 (with an addition at the end of 8.2) into ""coap://[2001:db8:f0::1]/.well-known/core"".

Because ""/temp"" starts with a single slash, the record's target is resolved by replacing the path ""/.well-known/core"" from the Base URI (section 5.2 [RFC3986]) with the relative target URI ""/temp"" into ""coap://[2001:db8:f0::1]/temp"".

B.1.2. Interpreting attributes and relations

Some more information but the record's target can be obtained from the payload: the resource type of the target is "temperature", and its content format is text/plain (ct=0).

A relation in a web link is a three-part statement that specifies a named relation between the so-called "context resource" and the target resource, like "_This page_ has _its table of contents_ at _/toc.html_". In link format documents, there is an implicit "host relation" specified with default parameter: rel="hosts".

In our example, the context resource of the link is the URI specified in the GET request "coap:://[2001:db8:f0::1]/.well-known/core". A full English expression of the "host relation" is:

'"coap://[2001:db8:f0::1]/.well-known/core" is hosting the resource "coap://[2001:db8:f0::1]/temp", which is of the resource type "temperature" and can be accessed using the text/plain content format.'

B.2. A slightly more complex example

Omitting the "rt=temperature" filter, the discovery query would have given some more records in the payload:

```
GET coap://[ff02::fd]:5683/.well-known/core

RES 2.05 Content
</temp>;rt=temperature;ct=0,
</light>;rt=light-lux;ct=0,
</t>;anchor="/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;anchor="/temp";
    rel="describedby"
```

Figure 31: Extended example of direct resource discovery

Parsing the third record, the client encounters the "anchor" parameter. It is a URI relative to the Base URI of the request and is thus resolved to ""coap://[2001:db8:f0::1]/sensors/temp"". That is the context resource of the link, so the "rel" statement is not about the target and the Base URI any more, but about the target and the resolved URI. Thus, the third record could be read as ""coap://[2001:db8:f0::1]/sensors/temp" has an alternate representation at "coap://[2001:db8:f0::1]/t"".

Following the same resolution steps, the fourth record can be read as ""coap://[2001:db8:f0::1]/sensors/temp" is described by "http://www.example.com/sensors/t123"".

B.3. Enter the Resource Directory

The resource directory tries to carry the semantics obtainable by classical CoAP discovery over to the resource lookup interface as faithfully as possible.

For the following queries, we will assume that the simple host has used Simple Registration to register at the resource directory that was announced to it, sending this request from its UDP port "[2001:db8:f0::1]:6553":

```
POST coap://[2001:db8:f01::ff]/.well-known/core?ep=simple-host1
```

Figure 32: Example request starting a simple registration

The resource directory would have accepted the registration, and queried the simple host's ".well-known/core" by itself. As a result, the host is registered as an endpoint in the RD with the name "simple-host1". The registration is active for 90000 seconds, and the endpoint registration Base URI is ""coap://[2001:db8:f0::1]"" following the resolution steps described in Appendix B.1.1. It should be remarked that the Base URI constructed that way always yields a URI of the form: scheme://authority without path suffix.

If the client now queries the RD as it would previously have issued a multicast request, it would go through the RD discovery steps by fetching "coap://[2001:db8:f0::ff]/.well-known/core?rt=core.rd-lookup-res", obtain "coap://[2001:db8:f0::ff]/rd-lookup/res" as the resource lookup endpoint, and issue a request to "coap://[2001:db8:f0::ff]/rd-lookup/res?rt=temperature" to receive the following data:

```
<coap://[2001:db8:f0::1]/temp>;rt=temperature;ct=0;
    anchor="coap://[2001:db8:f0::1]"
```

Figure 33: Example payload of a response to a resource lookup

This is not _literally_ the same response that it would have received from a multicast request, but it contains the equivalent statement:

’"coap://[2001:db8:f0::1]" is hosting the resource "coap://[2001:db8:f0::1]/temp", which is of the resource type "temperature" and can be accessed using the text/plain content format.’

(The difference is whether "/" or "/.well-known/core" hosts the resources, which does not matter in this application; if it did, the endpoint would have been more explicit. Actually, /.well-known/core does NOT host the resource but stores a URI reference to the resource.)

To complete the examples, the client could also query all resources hosted at the endpoint with the known endpoint name "simple-host1". A request to "coap://[2001:db8:f0::ff]/rd-lookup/res?ep=simple-host1" would return

```
<coap://[2001:db8:f0::1]/temp>;rt=temperature;ct=0;
    anchor="coap://[2001:db8:f0::1]",
<coap://[2001:db8:f0::1]/light>;rt=light-lux;ct=0;
    anchor="coap://[2001:db8:f0::1]",
<coap://[2001:db8:f0::1]/t>;
    anchor="coap://[2001:db8:f0::1]/sensors/temp";rel=alternate,
<http://www.example.com/sensors/t123>;
    anchor="coap://[2001:db8:f0::1]/sensors/temp";rel="describedby"
```

Figure 34: Extended example payload of a response to a resource lookup

All the target and anchor references are already in absolute form there, which don’t need to be resolved any further.

Had the simple host done an equivalent full registration with a base= parameter (e.g. "?ep=simple-host1&base=coap+tcp://simple-host1.example.com"), that context would have been used to resolve the relative anchor values instead, giving

```
<coap+tcp://simple-host1.example.com/temp>;rt=temperature;ct=0;
    anchor="coap+tcp://simple-host1.example.com"
```

Figure 35: Example payload of a response to a resource lookup with a dedicated base URI

and analogous records.

B.4. A note on differences between link-format and Link headers

While link-format and Link headers look very similar and are based on the same model of typed links, there are some differences between [RFC6690] and [RFC8288], which are dealt with differently:

- "Resolving the target against the anchor": [RFC6690] Section 2.1 states that the anchor of a link is used as the Base URI against which the term inside the angle brackets (the target) is resolved, falling back to the resource's URI with paths stripped off (its "Origin"). In contrast to that, [RFC8288] Section B.2 describes that the anchor is immaterial to the resolution of the target reference.

  RFC6690, in the same section, also states that absent anchors set the context of the link to the target's URI with its path stripped off, while according to [RFC8288] Section 3.2, the context is the resource's base URI.

  The rules introduced in Appendix C ensure that an RD does not need to deal with those differences when processing input data. Lookup results are required to be absolute references for the same reason.

- There is no percent encoding in link-format documents.

  A link-format document is a UTF-8 encoded string of Unicode characters and does not have percent encoding, while Link headers are practically ASCII strings that use percent encoding for non-ASCII characters, stating the encoding explicitly when required.

  For example, while a Link header in a page about a Swedish city might read

  "Link: </temperature/Malm%C3%B6>;rel="live-environment-data""

a link-format document from the same source might describe the link as

"</temperature/Malmoe>;rel="live-environment-data""

Parsers and producers of link-format and header data need to be aware of this difference.

Appendix C. Limited Link Format

The CoRE Link Format as described in [RFC6690] has been interpreted differently by implementers, and a strict implementation rules out some use cases of a Resource Directory (e.g. base values with path components).

This appendix describes a subset of link format documents called Limited Link Format. The rules herein are not very limiting in practice - all examples in RFC6690, and all deployments the authors are aware of already stick to them - but ease the implementation of resource directory servers.

It is applicable to representations in the application/link-format media type, and any other media types that inherit [RFC6690] Section 2.1.

A link format representation is in Limited Link format if, for each link in it, the following applies:

- 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),
- if the anchor reference starts with a scheme, the target reference starts with a scheme as well (i.e. relative references in target cannot be used when the anchor is a full URI), and
- the application does not care whether links without an explicitly given anchor have the origin's "/" or "/.well-known/core" resource as their link context.

Authors' Addresses

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Michael Koster
SmartThings
665 Clyde Avenue
Mountain View 94043
USA

Phone: +1-707-502-5136
Email: Michael.Koster@smartthings.com

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

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

Peter van der Stok
consultant

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

Christian Amsuess (editor)
Hollandstr. 12/4
1020
Austria

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

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 19, 2018

C. Jennings
Cisco
Z. Shelby
ARM
J. Arkko
A. Keranen
Ericsson
C. Bormann
Universitaet Bremen TZI

May 18, 2018

Sensor Measurement Lists (SenML)
draft-ietf-core-senml-16

Abstract

This specification defines a format for representing simple sensor measurements and device parameters in the Sensor Measurement Lists (SenML). Representations are defined in JavaScript Object Notation (JSON), Concise Binary Object Representation (CBOR), Extensible Markup Language (XML), and Efficient XML Interchange (EXI), which share the common SenML data model. A simple sensor, such as a temperature sensor, could use one of these media types in protocols such as HTTP or CoAP to transport the measurements of the sensor or to be configured.

Status of This Memo

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

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

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

This Internet-Draft will expire on November 19, 2018.

Table of Contents

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 it 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:

- 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.

- 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).

- 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

- 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 |fffff..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.l ...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 >> 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 .regexp  "[A-Zac-z0-9][-_:.A-Za-z0-9]*" / uint
b-label = tstr .regexp  "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

- Note 1: Assumed to be in WGS84 unless another reference frame is known for the sensor.

- 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 T0 = 275.15 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 the are common SI prefixes: Y, Z, E, P, T, G, M, k, h, da, d, c, n, u, p, f, a, z, y, Ki, Mi, Gi, Ti, Pi, Ei, Zi, Yi.

8. The following units should not be used as they are commonly used to represent other measurements Ky, Gal, dyn, etg, P, St, Mx, G, Oe, Gb, sb, Lmb, mph, Ci, R, RAD, REM, gal, bbl, qt, degF, Cal, BTU, HP, pH, B/s, psi, Torr, atm, at, bar, kWh.

9. The unit names are case sensitive and the correct case needs to be used, but symbols that differ only in case should not be allocated.

10. A number after a unit typically indicates the previous unit raised to that power, and the / indicates that the units that follow are the reciprocal. A unit should have only one / in the name.

11. A good list of common units can be found in the Unified Code for Units of Measure [UCUM].

12.2. SenML Label Registry

IANA will create a new registry for SenML labels. The initial content of the registry is:

Name	Label	CL	JSON Type	XML Type	EI	Reference
Base Name	bn	-2	String	string	a	RFC-AAAA
Base Time	bt	-3	Number	double	a	RFC-AAAA
Base Unit	bu	-4	String	string	a	RFC-AAAA
Base Value	bv	-5	Number	double	a	RFC-AAAA
Base Sum	bs	-6	Number	double	a	RFC-AAAA
Base Version	bver	-1	Number	int	a	RFC-AAAA
Name	n	0	String	string	a	RFC-AAAA
Unit	u	1	String	string	a	RFC-AAAA
Value	v	2	Number	double	a	RFC-AAAA
String Value	vs	3	String	string	a	RFC-AAAA
Boolean Value	vb	4	Boolean	boolean	a	RFC-AAAA
Data Value	vd	8	String	string	a	RFC-AAAA
Value Sum	s	5	Number	double	a	RFC-AAAA
Time	t	6	Number	double	a	RFC-AAAA
Update Time	ut	7	Number	double	a	RFC-AAAA

Table 7: IANA Registry for SenML Labels, CL = CBOR Label, EI = EXI ID

This is the same table as Table 1, with notes removed, and with columns added for the information that is all the same for this initial set of registrations, but will need to be supplied with a different value for new registrations.

All new entries must define the Label Name, Label, and XML Type but the CBOR labels SHOULD be left empty as CBOR will use the string encoding for any new labels. The EI column contains the EXI schemaId value of the first Schema which includes this label or is empty if this label was not intended for use with EXI. The Note field SHOULD contain information about where to find out more information about this label.

The JSON, CBOR, and EXI types are derived from the XML type. All XML numeric types such as double, float, integer and int become a JSON Number. XML boolean and string become a JSON Boolean and String respectively. CBOR represents numeric values with a CBOR type that does not lose any information from the JSON value. EXI uses the XML types.

New entries can be added to the registration by Expert Review as defined in [RFC8126]. Experts should exercise their own good judgment but need to consider that shorter labels should have more strict review. New entries should not be made that counteract the advice at the end of Section 4.5.4.

All new SenML labels that have "base" semantics (see Section 4.1) MUST start with the character 'b'. Regular labels MUST NOT start with that character. All new SenML labels with Value semantics (see Section 4.2) MUST have "Value" in their (long form) name.

Extensions that add a label that is intended for use with XML need to create a new RelaxNG scheme that includes all the labels in the IANA registry.

Extensions that add a label that is intended for use with EXI need to create a new XSD Schema that includes all the labels in the IANA registry and then allocate a new EXI schemaId value. Moving to the next letter in the alphabet is the suggested way to create the new value for the EXI schemaId. Any labels with previously blank ID values SHOULD be updated in the IANA table to have their ID set to this new schemaId value.

Extensions that are mandatory to understand to correctly process the Pack MUST have a label name that ends with the '_' character.

12.3. Media Type Registrations

The following registrations are done following the procedure specified in [RFC6838] and [RFC7303]. This document registers media types for each serialization format of SenML (JSON, CBOR, XML, and EXI) and also a corresponding set of media types for the streaming use (SensML, see Section 4.8). Clipboard formats are defined for the JSON and XML forms of SenML but not for streams or non-textual formats.

The reason there are both SenML and the streaming SensML formats is that they are not the same data formats and they require separate negotiation to understand if they are supported and which one is being used. The non streaming format is required to have some sort of end of pack syntax which indicates there will be no more records. Many implementations that receive SenML wait for this end of pack marker before processing any of the records. On the other hand, with the streaming formats, it is explicitly not required to wait for this end of pack marker. Many implementations that produce streaming SensML will never send this end of pack marker so implementations that receive streaming SensML can not wait for the end of pack marker before they start processing the records. Given the SenML and

streaming SenML are different data formats, and the requirement for separate negotiation, a media type for each one is needed.

Note to RFC Editor - please remove this paragraph. Note that a request for media type review for senml+json was sent to the media-types@iana.org on Sept 21, 2010. A second request for all the types was sent on October 31, 2016. Please change all instances of RFC-AAAA with the RFC number of this document.

12.3.1. senml+json Media Type Registration

Type name: application

Subtype name: senml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC8259]. See RFC-AAAA for details. This simplifies implementation of very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any JSON key value pairs that they do not understand unless the key ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the JSON object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+json is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senml

Windows Clipboard Name: "JSON Sensor Measurement List"

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-json conforms to public.text

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.2. sensml+json Media Type Registration

Type name: application

Subtype name: sensml+json

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using a subset of the encoding allowed in [RFC8259]. See RFC-AAAA for details. This simplifies implementation of very simple system and does not impose any significant limitations as all this data is meant for machine to machine communications and is not meant to be human readable.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any JSON key value pairs that they do not understand unless the key ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the JSON object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/sensml+json is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensml

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.3. senml+cbor Media Type Registration

Type name: application

Subtype name: senml+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC7049]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any key value pairs that they do not understand unless the key ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver"

field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the CBOR object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+cbor is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senmlc

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-cbor conforms to public.data

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.4. sensml+cbor Media Type Registration

Type name: application

Subtype name: sensml+cbor

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [RFC7049]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any key value pairs that they do not understand unless the key ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" field can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the CBOR object.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/sensml+cbor is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensmlc

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.5. senml+xml Media Type Registration

Type name: application

Subtype name: senml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-xml-20081126]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml+xml is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senmlx

Windows Clipboard Name: "XML Sensor Measurement List"

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-xml conforms to public.xml

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.6. sensml+xml Media Type Registration

Type name: application

Subtype name: sensml+xml

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-xml-20081126]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/sensml+xml is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensmlx

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.7. senml-exi Media Type Registration

Type name: application

Subtype name: senml-exi

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-exi-20140211]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack. Further information on using schemas to guide the EXI can be found in RFC-AAAA.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/senml-exi is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): senmle

Macintosh file type code(s): none

Macintosh Universal Type Identifier code: org.ietf.senml-exi conforms to public.data

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.3.8. sensml-exi Media Type Registration

Type name: application

Subtype name: sensml-exi

Required parameters: none

Optional parameters: none

Encoding considerations: Must be encoded as using [W3C.REC-exi-20140211]. See RFC-AAAA for details.

Security considerations: See Section 13 of RFC-AAAA.

Interoperability considerations: Applications MUST ignore any XML tags or attributes that they do not understand unless the attribute name ends with the '_' character in which case an error MUST be generated. This allows backwards compatible extensions to this specification. The "bver" attribute in the senml XML tag can be used to ensure the receiver supports a minimal level of functionality needed by the creator of the XML SenML Pack. Further information on using schemas to guide the EXI can be found in RFC-AAAA.

Published specification: RFC-AAAA

Applications that use this media type: The type is used by systems that report e.g., electrical power usage and environmental information such as temperature and humidity. It can be used for a wide range of sensor reporting systems.

Fragment identifier considerations: Fragment identification for application/sensml-exi is supported by using fragment identifiers as specified by RFC-AAAA.

Additional information:

Magic number(s): none

File extension(s): sensmle

Macintosh file type code(s): none

Person & email address to contact for further information: Cullen Jennings <fluffy@iii.ca>

Intended usage: COMMON

Restrictions on usage: None

Author: Cullen Jennings <fluffy@iii.ca>

Change controller: IESG

12.4. XML Namespace Registration

This document registers the following XML namespaces in the IETF XML registry defined in [RFC3688].

URI: urn:ietf:params:xml:ns:senml

Registrant Contact: The IESG.

XML: N/A, the requested URIs are XML namespaces

12.5. CoAP Content-Format Registration

IANA is requested to assign CoAP Content-Format IDs for the SenML media types in the "CoAP Content-Formats" sub-registry, within the "CoRE Parameters" registry [RFC7252]. IDs for the JSON, CBOR, and EXI Content-Formats are assigned from the "Expert Review" (0-255) range and for the XML Content-Format from the "IETF Review or IESG Approval" range. The assigned IDs are shown in Table 8.

Media type	Encoding	ID	Reference
application/senml+json	-	TBD:110	RFC-AAAA
application/sensml+json	-	TBD:111	RFC-AAAA
application/senml+cbor	-	TBD:112	RFC-AAAA
application/sensml+cbor	-	TBD:113	RFC-AAAA
application/senml-exi	-	TBD:114	RFC-AAAA
application/sensml-exi	-	TBD:115	RFC-AAAA
application/senml+xml	-	TBD:310	RFC-AAAA
application/sensml+xml	-	TBD:311	RFC-AAAA

Table 8: CoAP Content-Format IDs

13. Security Considerations

Sensor data presented with SenML can contain a wide range of information ranging from information that is very public, such as the outside temperature in a given city, to very private information that requires integrity and confidentiality protection, such as patient health information. When SenML is used for configuration or actuation, it can be used to change the state of systems and also impact the physical world, e.g., by turning off a heater or opening a lock.

The SenML formats alone do not provide any security and instead rely on the protocol that carries them to provide security. Applications using SenML need to look at the overall context of how these formats will be used to decide if the security is adequate. In particular for sensitive sensor data and actuation use it is important to ensure that proper security mechanisms are used to provide, e.g., confidentiality, data integrity, and authentication as appropriate for the usage.

The SenML formats defined by this specification do not contain any executable content. However, future extensions could potentially embed application specific executable content in the data.

SenML Records are intended to be interpreted in the context of any applicable base values. If records become separated from the record that establishes the base values, the data will be useless or, worse, wrong. Care needs to be taken in keeping the integrity of a Pack that contains unresolved SenML Records (see Section 4.6).

See also Section 14.

14. Privacy Considerations

Sensor data can range from information with almost no privacy considerations, such as the current temperature in a given city, to highly sensitive medical or location data. This specification provides no security protection for the data but is meant to be used inside another container or transfer protocol such as S/MIME [RFC5751] or HTTP with TLS [RFC2818] that can provide integrity, confidentiality, and authentication information about the source of the data.

The name fields need to uniquely identify the sources or destinations of the values in a SenML Pack. However, the use of long-term stable unique identifiers can be problematic for privacy reasons [RFC6973], depending on the application and the potential of these identifiers to be used in correlation with other information. They should be used with care or avoided as for example described for IPv6 addresses in [RFC7721].

15. Acknowledgement

We would like to thank Alexander Pelov, Alexey Melnikov, Andrew McClure, Andrew McGregor, Bjoern Hoehrmann, Christian Amsuess, Christian Groves, Daniel Peintner, Jan-Piet Mens, Jim Schaad, Joe Hildebrand, John Klensin, Karl Palsson, Lennart Duhrsen, Lisa Dusseault, Lyndsay Campbell, Martin Thomson, Michael Koster, Peter Saint-Andre, Roni Even, and Stephen Farrell, for their review comments.

16. References

16.1. Normative References

[BIPM] Bureau International des Poids et Mesures, "The International System of Units (SI)", 8th edition, 2006.

[IEEE.754.1985] Institute of Electrical and Electronics Engineers, "Standard for Binary Floating-Point Arithmetic", IEEE Standard 754, August 1985.

[NIST811] Thompson, A. and B. Taylor, "Guide for the Use of the International System of Units (SI)", NIST Special Publication 811, 2008.

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

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, DOI 10.17487/RFC3629, November 2003, <https://www.rfc-editor.org/info/rfc3629>.

[RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688, DOI 10.17487/RFC3688, January 2004, <https://www.rfc-editor.org/info/rfc3688>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <https://www.rfc-editor.org/info/rfc6838>.

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

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <https://www.rfc-editor.org/info/rfc7252>.

[RFC7303] Thompson, H. and C. Lilley, "XML Media Types", RFC 7303, DOI 10.17487/RFC7303, July 2014, <https://www.rfc-editor.org/info/rfc7303>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

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

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RNC] ISO/IEC, "Information technology -- Document Schema Definition Language (DSDL) -- Part 2: Regular-grammar-based validation -- RELAX NG", ISO/IEC 19757-2, Annex C: RELAX NG Compact syntax, December 2008.

[TIME_T] The Open Group Base Specifications, "Vol. 1: Base Definitions, Issue 7", Section 4.15 'Seconds Since the Epoch', IEEE Std 1003.1, 2013 Edition, 2013, <http://pubs.opengroup.org/onlinepubs/9699919799/basedefs/V1_chap04.html#tag_04_15>.

[W3C.REC-exi-20140211] Schneider, J., Kamiya, T., Peintner, D., and R. Kyusakov, "Efficient XML Interchange (EXI) Format 1.0 (Second Edition)", World Wide Web Consortium Recommendation REC-exi-20140211, February 2014, <http://www.w3.org/TR/2014/REC-exi-20140211>.

[W3C.REC-xml-20081126] Bray, T., Paoli, J., Sperberg-McQueen, M., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", World Wide Web Consortium Recommendation REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126>.

[W3C.REC-xmlschema-1-20041028] Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-1-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-1-20041028>.

[XPointerElement] Grosso, P., Maler, E., Marsh, J., and N. Walsh, "XPointer element() Scheme", W3C Recommendation REC-xptr-element, March 2003, <https://www.w3.org/TR/2003/REC-xptr-element-20030325/>.

[XPointerFramework] Grosso, P., Maler, E., Marsh, J., and N. Walsh, "XPointer Framework", W3C Recommendation REC-XPointer-Framework, March 2003, <http://www.w3.org/TR/2003/REC-xptr-framework-20030325/>.

16.2. Informative References

[AN1796] Linke, B., "Overview of 1-Wire Technology and Its Use", June 2008, <http://pdfserv.maximintegrated.com/en/an/AN1796.pdf>.

[I-D.ietf-cbor-cddl] Birkholz, H., Vigano, C., and C. Bormann, "Concise data definition language (CDDL): a notational convention to express CBOR data structures", draft-ietf-cbor-cddl-02 (work in progress), February 2018.

[I-D.ietf-core-dev-urn] Arkko, J., Jennings, C., and Z. Shelby, "Uniform Resource Names for Device Identifiers", draft-ietf-core-dev-urn-01 (work in progress), March 2018.

[I-D.ietf-core-interfaces] Shelby, Z., Vial, M., Koster, M., Groves, C., Zhu, J., and B. Silverajan, "Reusable Interface Definitions for Constrained RESTful Environments", draft-ietf-core-interfaces-11 (work in progress), March 2018.

[IEEE802.1as-2011] IEEE, "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks", 2011.

[IEEE802.1ba-2011] IEEE, "IEEE Standard for Local and metropolitan area networks--Audio Video Bridging (AVB) Systems", 2011.

[ISO-80000-5] "Quantities and units - Part 5: Thermodynamics", ISO 80000-5, Edition 1.0, May 2007.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/info/rfc2818>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/info/rfc3986>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/info/rfc4122>.

[RFC4151] Kindberg, T. and S. Hawke, "The 'tag' URI Scheme", RFC 4151, DOI 10.17487/RFC4151, October 2005, <https://www.rfc-editor.org/info/rfc4151>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, DOI 10.17487/RFC5751, January 2010, <https://www.rfc-editor.org/info/rfc5751>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, DOI 10.17487/RFC5952, August 2010, <https://www.rfc-editor.org/info/rfc5952>.

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

[RFC6920] Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", RFC 6920, DOI 10.17487/RFC6920, April 2013, <https://www.rfc-editor.org/info/rfc6920>.

[RFC6973] Cooper, A., Tschofenig, H., Aboba, B., Peterson, J., Morris, J., Hansen, M., and R. Smith, "Privacy Considerations for Internet Protocols", RFC 6973, DOI 10.17487/RFC6973, July 2013, <https://www.rfc-editor.org/info/rfc6973>.

[RFC7111] Hausenblas, M., Wilde, E., and J. Tennison, "URI Fragment Identifiers for the text/csv Media Type", RFC 7111, DOI 10.17487/RFC7111, January 2014, <https://www.rfc-editor.org/info/rfc7111>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[RFC7721] Cooper, A., Gont, F., and D. Thaler, "Security and Privacy Considerations for IPv6 Address Generation Mechanisms", RFC 7721, DOI 10.17487/RFC7721, March 2016, <https://www.rfc-editor.org/info/rfc7721>.

[RFC8141] Saint-Andre, P. and J. Klensin, "Uniform Resource Names (URNs)", RFC 8141, DOI 10.17487/RFC8141, April 2017, <https://www.rfc-editor.org/info/rfc8141>.

[UCUM] Schadow, G. and C. McDonald, "The Unified Code for Units of Measure (UCUM)", Regenstrief Institute and Indiana University School of Informatics, 2013, <http://unitsofmeasure.org/ucum.html>.

Authors' Addresses

Cullen Jennings
Cisco
400 3rd Avenue SW
Calgary, AB T2P 4H2
Canada

Email: fluffy@iii.ca

Zach Shelby
ARM
150 Rose Orchard
San Jose 95134
USA

Phone: +1-408-203-9434
Email: zach.shelby@arm.com

Jari Arkko
Ericsson
Jorvas 02420
Finland

Email: jari.arkko@piuha.net

Ari Keranen
Ericsson
Jorvas 02420
Finland

Email: ari.keranen@ericsson.com

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

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

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: January 9, 2020

M. Veillette, Ed.
Trilliant Networks Inc.
A. Pelov, Ed.
I. Petrov, Ed.
Acklio

July 08, 2019

YANG Schema Item iDentifier (SID)
draft-ietf-core-sid-07

Abstract

YANG Schema Item iDentifiers (SID) are globally unique 64-bit unsigned integers used to identify YANG items. This document defines the semantics, the registration, and assignment processes of SIDs. To enable the implementation of these processes, this document also defines a file format used to persist and publish assigned SIDs.

Status of This Memo

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

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

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

This Internet-Draft will expire on January 9, 2020.

Copyright Notice

Table of Contents

1. Introduction

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both NETCONF [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using names. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG items is required. This compact identifier, called SID, is encoded using a 64-bit unsigned integer. The following items are identified using SIDs:

- identities
- data nodes (Note: including those parts of a YANG template as defined by the 'yang-data' extension.)
- RPCs and associated input(s) and output(s)
- actions and associated input(s) and output(s)
- notifications and associated information
- YANG modules, submodules and features

SIDs are globally unique integers, a registration system is used in order to guarantee their uniqueness. SIDs are registered in blocks called "SID ranges".

Assignment of SIDs to YANG items can be automated. For more details how this could be achieved, please consult Appendix B.

SIDs are assigned permanently, items introduced by a new revision of a YANG module are added to the list of SIDs already assigned.

Section 3 provides more details about the registration process of YANG modules and associated SIDs. To enable the implementation of this registry, Section 4 defines a standard file format used to store and publish SIDs.

2. Terminology and Notation

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

The following terms are defined in [RFC7950]:

- 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, a submodule or a feature defined using the YANG modeling language.
- path: A path is a string that identifies a schema node within the schema tree. A path consists of the list of schema node identifier(s) separated by slashes ("/"). Schema node identifier(s) are always listed from the top-level schema node up to the targeted schema node. (e.g. "/ietf-system:system-state/clock/current-datetime")
- YANG Schema Item iDentifier (SID): Unsigned integer used to identify different YANG items.

3. ".sid" file lifecycle

YANG is a language designed to model data accessed using one of the compatible protocols (e.g. NETCONF [RFC6241], RESCONF [RFC8040] and CoRECONF [I-D.ietf-core-comi]). A YANG module defines hierarchies of data, including configuration, state data, RPCs, actions and notifications.

Many YANG modules are not created in the context of constrained applications. YANG modules can be implemented using NETCONF [RFC6241] or RESTCONF [RFC8040] without the need to assign SIDs.

As needed, authors of YANG modules can assign SIDs to their YANG modules. In order to do that, they should first obtain a SID range from a registry and use that range to assign or generate SIDs to items of their YANG module. For example how this could be achieved, please refer to Appendix C.

Registration of the .sid file associated to a YANG module is optional but recommended to promote interoperability between devices and to avoid duplicate allocation of SIDs to a single YANG module. Different registries might have different requirements for the registration and publication of the ".sid" files. For diagram of one

of the possibilities, please refer to the activity diagram on Figure 1 in Appendix C.

Each time a YANG module or one of its imported module(s) or included sub-module(s) is updated, the ".sid" file MAY need to be updated. This update SHOULD also be performed using an automated tool.

If a new revision requires more SIDs than initially allocated, a new SID range MUST be added to the 'assignment-ranges' as defined in Section 4. These extra SIDs are used for subsequent assignments.

For an example of this update process, see activity diagram Figure 2 in Appendix C.

4. ".sid" file format

".sid" files are used to persist and publish SIDs assigned to the different YANG items of a specific YANG module. The following YANG module defined the structure of this file, encoding is performed using the rules defined in [RFC7951].

```
<CODE BEGINS> file "ietf-sid-file@2017-11-26.yang"
module ietf-sid-file {
  namespace "urn:ietf:params:xml:ns:yang:ietf-sid-file";
  prefix sid;

  import ietf-yang-types {
    prefix yang;
  }

  organization
    "IETF Core Working Group";

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

     Andy Bierman
     <mailto:andy@yumaworks.com>

     Alexander Pelov
     <mailto:a@ackl.io>";

  description
    "This module defines the structure of the .sid files.

     Each .sid file contains the mapping between the different
     string identifiers defined by a YANG module and a
```

```
        corresponding numeric value called SID.";

    revision 2017-11-26 {
      description
        "Initial revision.";
      reference
        "[I-D.ietf-core-sid] YANG Schema Item iDentifier (SID)";
    }

    typedef revision-identifier {
      type string {
        pattern '\d{4}-\d{2}-\d{2}';
      }
      description
        "Represents a date in YYYY-MM-DD format.";
    }

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

    typedef schema-node-path {
      type string {
        pattern
          '/[a-zA-Z_][a-zA-Z0-9\-_.]*:[a-zA-Z_][a-zA-Z0-9\-_.]*' +
          '(/[a-zA-Z_][a-zA-Z0-9\-_.]*(:[a-zA-Z_][a-zA-Z0-9\-_.]*)?)*';
      }
      description
        "Identifies a schema-node path string for use in the
         SID registry. This string format follows the rules
         for an instance-identifier, as defined in RFC 7959,
         except that no predicates are allowed.

         This format is intended to support the YANG 1.1 ABNF
         for a schema node identifier, except module names
         are used instead of prefixes, as specified in RFC 7951.";
      reference
        "RFC 7950, The YANG 1.1 Data Modeling Language;
         Section 6.5: Schema Node Identifier;
         RFC 7951, JSON Encoding of YANG Data;
         Section 6.11: The instance-identifier type";
    }

    leaf module-name {
```

```
      type yang:yang-identifier;
      description
        "Name of the YANG module associated with this .sid file.";
    }

    leaf module-revision {
      type revision-identifier;
      description
        "Revision of the YANG module associated with this .sid file.
         This leaf is not present if no revision statement is
         defined in the YANG module.";
    }

    list assigment-ranges {
      key "entry-point";
      description
        "SID range(s) allocated to the YANG module identified by
         'module-name' and 'module-revision'.";

      leaf entry-point {
        type sid;
        mandatory true;
        description
          "Lowest SID available for assignment.";
      }

      leaf size {
        type uint64;
        mandatory true;
        description
          "Number of SIDs available for assignment.";
      }
    }

    list items {
      key "namespace identifier";
      description
        "Each entry within this list defined the mapping between
         a YANG item string identifier and a SID. This list MUST
         include a mapping entry for each YANG item defined by
         the YANG module identified by 'module-name' and
         'module-revision'.";

      leaf namespace {
        type enumeration {
          enum module {
            value 0;
            description
```

```
              "All module and submodule names share the same
               global module identifier namespace.";
            }
            enum identity {
              value 1;
              description
                "All identity names defined in a module and its
                 submodules share the same identity identifier
                 namespace.";
            }
            enum feature {
              value 2;
              description
                "All feature names defined in a module and its
                 submodules share the same feature identifier
                 namespace.";
            }
            enum data {
              value 3;
              description
                "The namespace for all data nodes, as defined in YANG.";
            }
          }
          description
            "Namespace of the YANG item for this mapping entry.";
        }

        leaf identifier {
          type union {
            type yang:yang-identifier;
            type schema-node-path;
          }
          description
            "String identifier of the YANG item for this mapping entry.

             If the corresponding 'namespace' field is 'module',
             'feature', or 'identity', then this field MUST
             contain a valid YANG identifier string.

             If the corresponding 'namespace' field is 'data',
             then this field MUST contain a valid schema node
             path.";
         }

        leaf sid {
          type sid;
          mandatory true;
          description
```

```
            "SID assigned to the YANG item for this mapping entry.";
        }
      }
    }
    <CODE ENDS>
```

5. Security Considerations

This document defines a new type of identifier used to encode data models defined in YANG [RFC7950]. As such, this identifier does not contribute to any new security issues in addition of those identified for the specific protocols or contexts for which it is used.

6. IANA Considerations

6.1. Register SID File Format Module

This document registers one YANG module in the "YANG Module Names" registry [RFC6020]:

- name: ietf-sid-file
- namespace: urn:ietf:params:xml:ns:yang:ietf-sid-file
- prefix: sid
- reference: [[THISRFC]]

6.2. Create new IANA Registry: "SID Mega-Range" registry

The name of this registry is "SID Mega-Range". This registry is used to record the delegation of the management of a block of SIDs to third parties (such as SDOs or registrars).

6.2.1. Structure

Each entry in this registry must include:

- The entry point (first SID) of the registered SID block.
- The size of the registered SID block. The size MUST be one million (1 000 000) SIDs.
- The contact information of the requesting organization including:
  - The policy of SID range allocations: Public, Private or Both.
  - Organization name

* URL

The information associated to the Organization name should not be publicly visible in the registry, but should be available. This information includes contact email and phone number and change controller email and phone number.

6.2.2. Allocation policy

The IANA policy for future additions to this registry is "Expert Review" [RFC8126].

An organization requesting to manage a SID Range (and thus have an entry in the SID Mega-Range Registry), must ensure the following capacities:

- The capacity to manage and operate a SID Range Registry. A SID Range Registry MUST provide the following information for all SID Ranges allocated by the Registry:
  - Entry Point of allocated SID Range
  - Size of allocated SID Range
  - Type: Public or Private
    - Public Ranges MUST include at least a reference to the YANG module and ".sid" files for that SID Range.
    - Private Ranges MUST be marked as "Private"
- A Policy of allocation, which clearly identifies if the 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.

6.2.2.1. First allocation

For a first allocation to be provided, the requesting organization must demonstrate a functional registry infrastructure.

6.2.2.2. Consecutive allocations

On subsequent allocation request(s), the organization must demonstrate the exhaustion of the prior range. These conditions need to be asserted by the assigned expert(s).

If that extra-allocation is done within 3 years from the last allocation, the experts need to discuss this request on the CORE working group mailing list and consensus needs to be obtained before allocating a new Mega-Range.

6.2.3. Initial contents of the Registry

The initial entry in this registry is allocated to IANA:

Entry Point	Size	Allocation	Organization name	URL
0	1000000	Public	IANA	iana.org

6.3. Create a new IANA Registry: IETF SID Range Registry (managed by IANA)

6.3.1. Structure

Each entry in this registry must include:

- The SID range entry point.
- The SID range size.
- The YANG module name.
- Document reference.

6.3.2. Allocation policy

The first million SIDs assigned to IANA is sub-divided as follows:

- The range of 0 to 999 (size 1000) is "Reserved" as defined in [RFC8126].
- The range of 1000 to 59,999 (size 59,000) is reserved for YANG modules defined in RFCs. The IANA policy for additions to this registry is "Expert Review" [RFC8126].

  * The Expert MUST verify that the YANG module for which this allocation is made has an RFC (existing RFC) OR is on track to become RFC (early allocation with a request from the WG chairs).

- The SID range allocated for a YANG module can follow in one of the four categories:
  * SMALL (50 SIDs)
  * MEDIUM (100 SIDs)
  * LARGE (250 SIDs)
  * CUSTOM (requested by the YANG module author, with a maximum of 1000 SIDs). In all cases, the size of a SID range assigned to a YANG module should be at least 33% above the current number of YANG items. This headroom allows assignment within the same range of new YANG items introduced by subsequent revisions. A larger SID range size may be requested by the authors if this recommendation is considered insufficient. It is important to note that an additional SID range can be allocated to an existing YANG module if the initial range is exhausted.
- 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	Reserved
1,000	59,000	Expert Review
60,000	40,000	Experimental use
100,000	900,000	Reserved

6.3.3. Initial contents of the registry

Initial entries in this registry are as follows:

Entry Point	Size	Module name	Document reference
1000	100	ietf-comi	[I-D.ietf-core-comi]
1100	50	ietf-yang-types	[RFC6991]
1150	50	ietf-inet-types	[RFC6991]
1200	50	iana-crypt-hash	[RFC7317]
1250	50	ietf-netconf-acm	[RFC8341]
1300	50	ietf-sid-file	RFCXXXX
1500	100	ietf-interfaces	[RFC8343]
1600	100	ietf-ip	[RFC8344]
1700	100	ietf-system	[RFC7317]
1800	400	iana-if-type	[RFC7224]

// RFC Ed.: replace XXXX with RFC number assigned to this draft.

For allocation, RFC publication of the YANG module is required as per [RFC8126]. The YANG module must be registered in the "YANG module Name" registry according to the rules specified in section 14 of [RFC6020].

6.4. Create new IANA Registry: "IETF SID Registry"

The name of this registry is "IETF SID Registry". This registry is used to record the allocation of SIDs for individual YANG module items.

6.4.1. Structure

Each entry in this registry must include:

- 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 number of actually allocated SIDs in the ".sid" file.

The ".sid" file is stored by IANA.

6.4.2. Allocation policy

The allocation policy is Expert review. The Expert MUST ensure that the following conditions are met:

- 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 SID Ranges for this YANG module (as allocated in the IETF SID Range Registry):
  - All SIDs in this ".sid" file MUST be within the ranges allocated to this YANG module in the "IETF SID Range Registry".
- If another ".sid" file has already allocated SIDs for this YANG module (e.g. for older or newer versions of the YANG module), the YANG items are assigned the same SIDs as in the the other ".sid" file.
- SIDs never change.

6.4.3. Initial contents of the registry

None.

7. Acknowledgments

The authors would like to thank Andy Bierman, Carsten Bormann, Abhinav Somaraju, Laurent Toutain, Randy Turner and Peter van der Stok for their help during the development of this document and their useful comments during the review process.

8. References

8.1. Normative References

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

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

[RFC7120] Cotton, M., "Early IANA Allocation of Standards Track Code Points", BCP 100, RFC 7120, DOI 10.17487/RFC7120, January 2014, <https://www.rfc-editor.org/info/rfc7120>.

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

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

8.2. Informative References

[I-D.ietf-core-comi] Veillette, M., Stok, P., Pelov, A., and A. Bierman, "CoAP Management Interface", draft-ietf-core-comi-05 (work in progress), May 2019.

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

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

[RFC6991] Schoenwaelder, J., Ed., "Common YANG Data Types", RFC 6991, DOI 10.17487/RFC6991, July 2013, <https://www.rfc-editor.org/info/rfc6991>.

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

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <https://www.rfc-editor.org/info/rfc7317>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8126] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://www.rfc-editor.org/info/rfc8126>.

[RFC8341] Bierman, A. and M. Bjorklund, "Network Configuration Access Control Model", STD 91, RFC 8341, DOI 10.17487/RFC8341, March 2018, <https://www.rfc-editor.org/info/rfc8341>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

[RFC8344] Bjorklund, M., "A YANG Data Model for IP Management", RFC 8344, DOI 10.17487/RFC8344, March 2018, <https://www.rfc-editor.org/info/rfc8344>.

Appendix A. ".sid" file example

The following .sid file (ietf-system@2014-08-06.sid) have been generated using the following yang modules:

- ietf-system@2014-08-06.yang
- ietf-yang-types@2013-07-15.yang
- ietf-inet-types@2013-07-15.yang
- ietf-netconf-acm@2012-02-22.yang
- iana-crypt-hash@2014-04-04.yang

```
{
  "assignment-ranges": [
    {
      "entry-point": 1700,
      "size": 100
    }
  ],
  "module-name": "ietf-system",
  "module-revision": "2014-08-06",
  "items": [
    {
      "namespace": "module",
      "identifier": "ietf-system",
      "sid": 1700
    },
```

```
{
  "namespace": "identity",
  "identifier": "authentication-method",
  "sid": 1701
},
{
  "namespace": "identity",
  "identifier": "local-users",
  "sid": 1702
},
{
  "namespace": "identity",
  "identifier": "radius",
  "sid": 1703
},
{
  "namespace": "identity",
  "identifier": "radius-authentication-type",
  "sid": 1704
},
{
  "namespace": "identity",
  "identifier": "radius-chap",
  "sid": 1705
},
{
  "namespace": "identity",
  "identifier": "radius-pap",
  "sid": 1706
},
{
  "namespace": "feature",
  "identifier": "authentication",
  "sid": 1707
},
{
  "namespace": "feature",
  "identifier": "dns-udp-tcp-port",
  "sid": 1708
},
{
  "namespace": "feature",
  "identifier": "local-users",
  "sid": 1709
},
{
  "namespace": "feature",
  "identifier": "ntp",
```

```
      "sid": 1710
    },
    {
      "namespace": "feature",
      "identifier": "ntp-udp-port",
      "sid": 1711
    },
    {
      "namespace": "feature",
      "identifier": "radius",
      "sid": 1712
    },
    {
      "namespace": "feature",
      "identifier": "radius-authentication",
      "sid": 1713
    },
    {
      "namespace": "feature",
      "identifier": "timezone-name",
      "sid": 1714
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:set-current-datetime",
      "sid": 1715
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:set-current-datetime/
                    current-datetime",
      "sid": 1716
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system",
      "sid": 1717
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system-restart",
      "sid": 1718
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system-shutdown",
      "sid": 1719
    },
```

```
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state",
        "sid": 1720
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/clock",
        "sid": 1721
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/clock/boot-datetime",
        "sid": 1722
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/clock/
                       current-datetime",
        "sid": 1723
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/platform",
        "sid": 1724
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/platform/machine",
        "sid": 1725
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/platform/os-name",
        "sid": 1726
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/platform/os-release",
        "sid": 1727
      },
      {
        "namespace": "data",
        "identifier": "/ietf-system:system-state/platform/os-version",
        "sid": 1728
      },
      {
        "namespace": "data",
```

```
      "identifier": "/ietf-system:system/authentication",
      "sid": 1729
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user",
      "sid": 1730
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/
                     user-authentication-order",
      "sid": 1731
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user/
                     authorized-key",
      "sid": 1732
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user/
                     authorized-key/algorithm",
      "sid": 1733
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user/
                     authorized-key/key-data",
      "sid": 1734
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user/
                     authorized-key/name",
      "sid": 1735
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user/
                     name",
      "sid": 1736
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/authentication/user/
                     password",
```

```
      "sid": 1737
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/clock",
      "sid": 1738
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/clock/timezone-name",
      "sid": 1739
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/clock/timezone-utc-offset",
      "sid": 1740
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/contact",
      "sid": 1741
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver",
      "sid": 1742
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/options",
      "sid": 1743
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/options/
                     attempts",
      "sid": 1744
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/options/
                     timeout",
      "sid": 1745
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/search",
      "sid": 1746
```

```
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server",
      "sid": 1747
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server/name",
      "sid": 1748
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server/
                     udp-and-tcp",
      "sid": 1749
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server/
                     udp-and-tcp/address",
      "sid": 1750
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/dns-resolver/server/
                     udp-and-tcp/port",
      "sid": 1751
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/hostname",
      "sid": 1752
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/location",
      "sid": 1753
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp",
      "sid": 1754
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/enabled",
      "sid": 1755
```

```
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server",
      "sid": 1756
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/
                     association-type",
      "sid": 1757
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/iburst",
      "sid": 1758
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/name",
      "sid": 1759
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/prefer",
      "sid": 1760
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/udp",
      "sid": 1761
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/udp/address",
      "sid": 1762
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/ntp/server/udp/port",
      "sid": 1763
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius",
      "sid": 1764
    },
    {
```

```
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/options",
      "sid": 1765
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/options/attempts",
      "sid": 1766
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/options/timeout",
      "sid": 1767
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/server",
      "sid": 1768
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/server/
                     authentication-type",
      "sid": 1769
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/server/name",
      "sid": 1770
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/server/udp",
      "sid": 1771
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/server/udp/
                     address",
      "sid": 1772
    },
    {
      "namespace": "data",
      "identifier": "/ietf-system:system/radius/server/udp/
                     authentication-port",
      "sid": 1773
    },
    {
```

```
        "namespace": "data",
        "identifier": "/ietf-system:system/radius/server/udp/
                       shared-secret",
        "sid": 1774
      }
    ]
  }
```

Appendix B. SID auto generation

Assignment of SIDs to YANG items can be automated, the recommended process to assign SIDs is as follows:

1. A tool extracts the different items defined for a specific YANG module.

2. The list of items is sorted in alphabetical order, 'namespace' in descending order, 'identifier' in ascending order. The 'namespace' and 'identifier' formats are described in the YANG module 'ietf-sid-file' defined in Section 4.

3. SIDs are assigned sequentially from the entry point up to the size of the registered SID range. This approach is recommended to minimize the serialization overhead, especially when delta between a reference SID and the current SID is used by protocols aiming to reduce message size.

4. If the number of items exceeds the SID range(s) allocated to a YANG module, an extra range is added for subsequent assignments.

Changes of SID files can also be automated using the same method described above, only unassigned YANG items are processed at step #3. Already existing items in the SID file should not be given new SIDs.

Appendix C. ".sid" file lifecycle

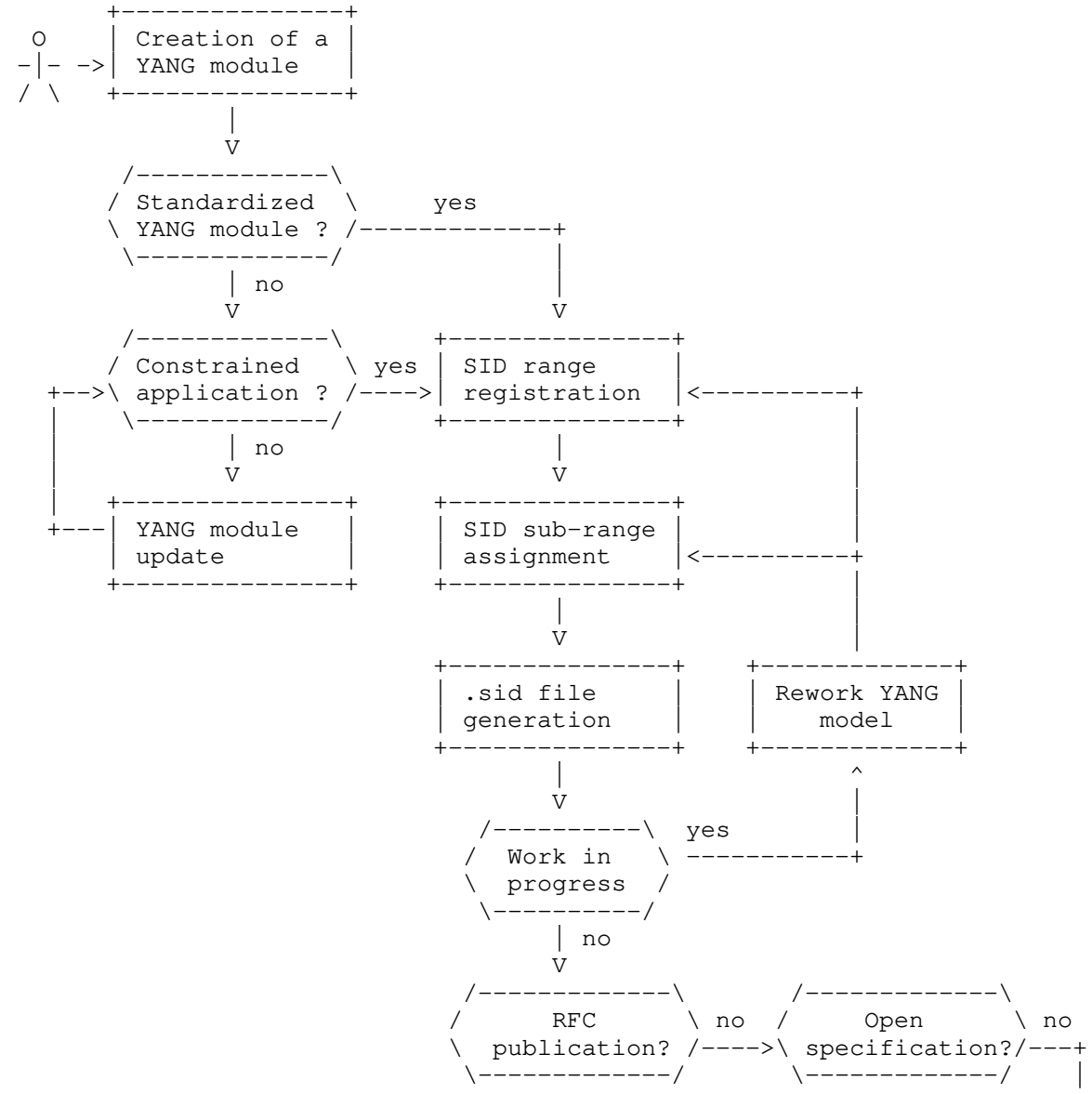
Before assigning SIDs to their YANG modules, YANG module authors must acquire a SID range from a "SID Range Registry". If the YANG module is part of an IETF draft or RFC, the SID range need to be acquired from the "IETF SID Range Registry" as defined in Section 6.3. For the other YANG modules, the authors can acquire a SID range from any "SID Range Registry" of their choice.

Once the SID range is acquired, the owner can use it to generate ".sid" file/s for his YANG module/s. It is recommended to leave some unallocated SIDs following the allocated range in each ".sid" file in order to allow better evolution of the YANG module in the future. Generation of ".sid" files should be performed using an automated

tool. Note that ".sid" files can only be generated for YANG modules and not for submodules.

C.1. SID File Creation

The following activity diagram summarizes the creation of a YANG module and its associated .sid file.



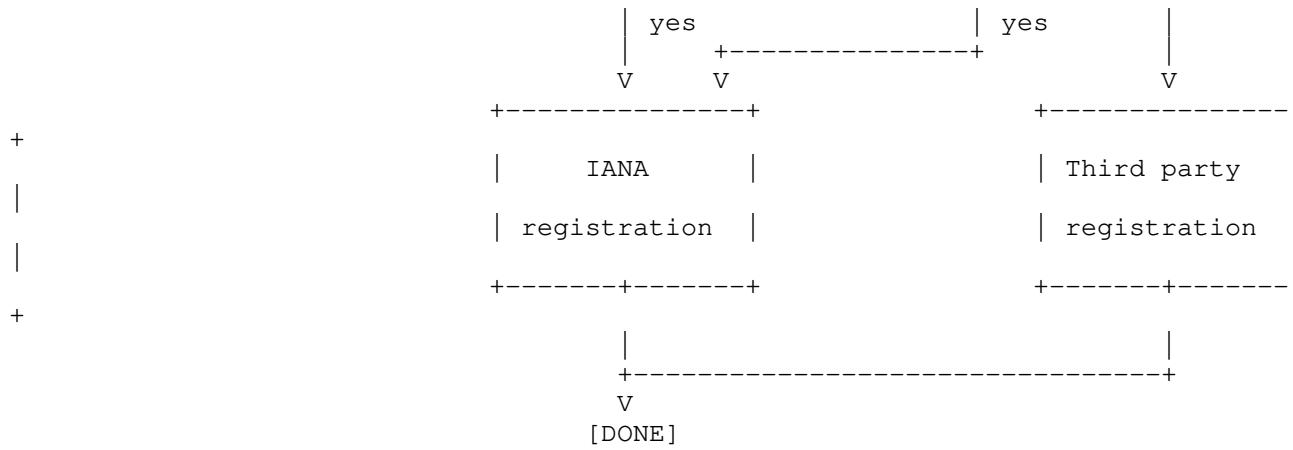


Figure 1: SID Lifecycle

C.2. SID File Update

The following Activity diagram summarizes the update of a YANG module and its associated .sid file.

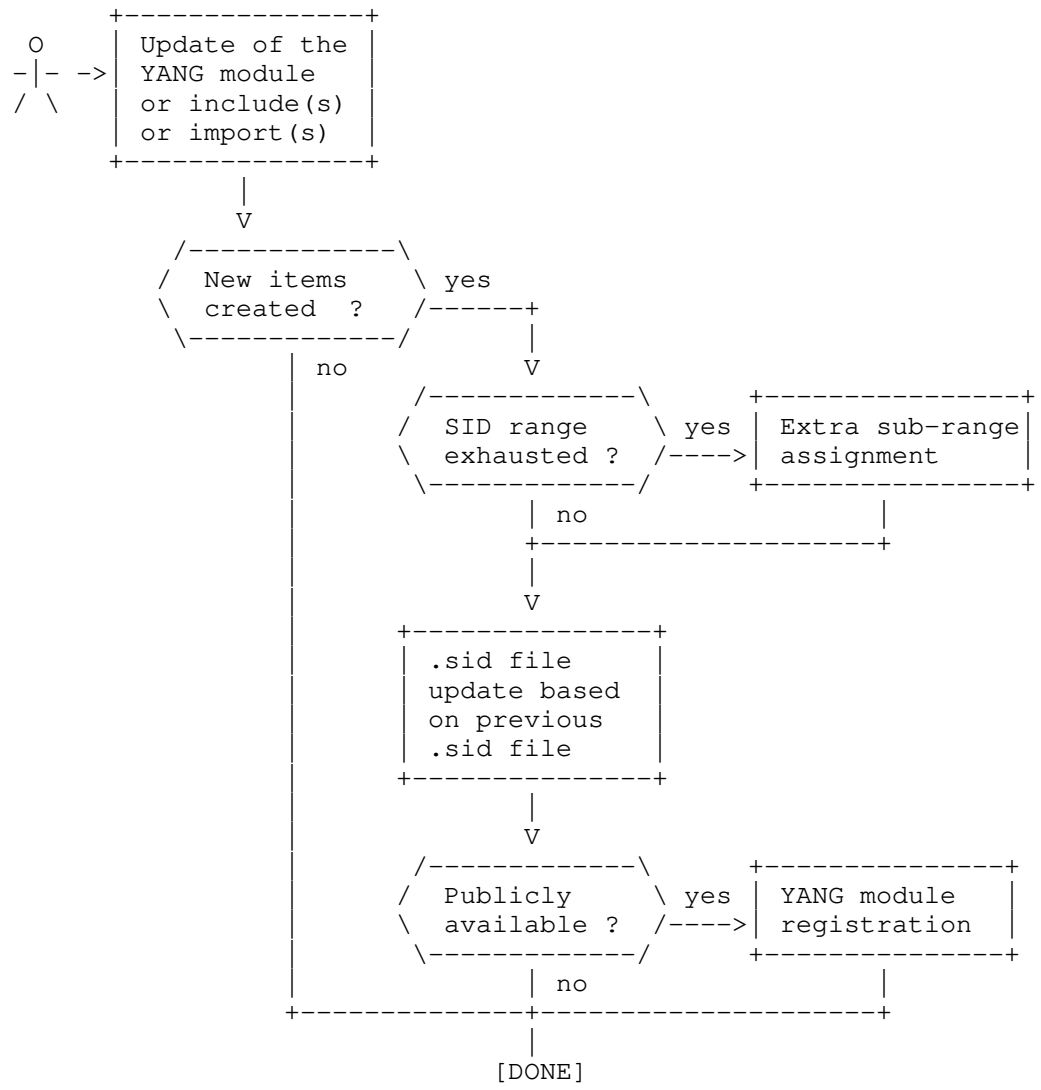


Figure 2: YANG and SID file update

Authors' Addresses

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

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

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

Email: a@ackl.io

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

Email: ivaylo@ackl.io

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: March 12, 2020

M. Veillette, Ed.
Trilliant Networks Inc.
I. Petrov, Ed.
A. Pelov
Acklio

September 09, 2019

CBOR Encoding of Data Modeled with YANG
draft-ietf-core-yang-cbor-11

Abstract

This document defines encoding rules for serializing configuration data, state data, RPC input and RPC output, Action input, Action output, notifications and yang data template defined within YANG modules using the Concise Binary Object Representation (CBOR) [RFC7049].

Status of This Memo

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

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

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

This Internet-Draft will expire on March 12, 2020.

Copyright Notice

Table of Contents

1. Introduction

The specification of the YANG 1.1 data modeling language [RFC7950] defines an XML encoding for data instances, i.e. contents of configuration datastores, state data, RPC inputs and outputs, action inputs and outputs, and event notifications.

A new set of encoding rules has been defined to allow the use of the same data models in environments based on the JavaScript Object Notation (JSON) Data Interchange Format [RFC8259]. This is accomplished in the JSON Encoding of Data Modeled with YANG specification [RFC7951].

The aim of this document is to define a set of encoding rules for the Concise Binary Object Representation (CBOR) [RFC7049]. The resulting encoding is more compact compared to XML and JSON and more suitable for Constrained Nodes and/or Constrained Networks as defined by [RFC7228].

2. Terminology and Notation

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

The following terms are defined in [RFC7950]:

- action
- anydata
- anyxml
- data node

- data tree
- datastore
- feature
- identity
- module
- notification
- RPC
- schema node
- schema tree
- submodule

The following terms are defined in [RFC8040]:

- yang-data (YANG extension)
- YANG data template

This specification also makes use of the following terminology:

- child: A schema node defined within a collection such as a container, a list, a case, a notification, an RPC input, an RPC output, an action input, an action output.
- delta: Difference between the current SID and a reference SID. A reference SID is defined for each context for which deltas are used.
- item: A schema node, an identity, a module, a submodule or a feature defined using the YANG modeling language.
- parent: The collection in which a schema node is defined.
- YANG Schema Item iDentifier (SID): Unsigned integer used to identify different YANG items.

3. Properties of the CBOR Encoding

This document defines CBOR encoding rules for YANG schema trees and their subtrees.

A collection such as container, list instance, notification, RPC input, RPC output, action input and action output is serialized using a CBOR map in which each child schema node is encoded using a key and a value. This specification supports two type of CBOR keys; YANG Schema Item iDentifier (SID) as defined in Section 3.2 and names as defined in Section 3.3. Each of these key types is encoded using a specific CBOR type which allows their interpretation during the deserialization process. Protocols or mechanisms implementing this specification can mandate the use of a specific key type.

In order to minimize the size of the encoded data, the proposed mapping avoids any unnecessary meta-information beyond those natively supported by CBOR. For instance, CBOR tags are used solely in the case of SID not encoded as delta, anyxml schema nodes and the union datatype to distinguish explicitly the use of different YANG datatypes encoded using the same CBOR major type.

Unless specified otherwise by the protocol or mechanism implementing this specification, the indefinite lengths encoding as defined in [RFC7049] section 2.2 SHALL be supported by CBOR decoders.

Data nodes implemented using a CBOR array, map, byte string, and text string can be instantiated but empty. In this case, they are encoded with a length of zero.

Application payloads carrying a value serialized using the rules defined by this specification (e.g. CoAP Content-Format) SHOULD include the identifier (e.g. SID, namespace qualified name, instance-identifier) of this value. When SIDs are used as identifiers, the reference SID SHALL be included in the payload to allow stateless conversion of delta values to SIDs.

Examples in section Section 4 include a root CBOR map with a single entry having a key set to either a namespace qualified name or a SID. This root CBOR map is provided only as a typical usage example and is not part of the present encoding rules. Only the value within this CBOR map is compulsory.

3.1. CBOR diagnostic notation

Within this document, CBOR binary contents are represented using an equivalent textual form called CBOR diagnostic notation as defined in [RFC7049] section 6. This notation is used strictly for

documentation purposes and is never used in the data serialization. Table 1 below provides a summary of this notation.

CBOR content	CBOR type	Diagnostic notation	Example	CBOR encoding
Unsigned integer	0	Decimal digits	123	18 7B
Negative integer	1	Decimal digits prefixed by a minus sign	-123	38 7A
Byte string	2	Hexadecimal value enclosed between single quotes and prefixed by an 'h'	h'F15C'	42 f15C
Text string	3	String of Unicode characters enclosed between double quotes	"txt"	63 747874
Array	4	Comma-separated list of values within square brackets	[1, 2]	82 01 02
Map	5	Comma-separated list of key : value pairs within curly braces	{ 1: 123, 2: 456 }	a2 01187B 021901C8
Boolean	7/20	false	false	F4
	7/21	true	true	F5
Null	7/22	null	null	F6
Not assigned	7/23	undefined	undefined	F7

Table 1: CBOR diagnostic notation summary

Note: CBOR binary contents shown in this specification are annotated with comments. These comments are delimited by slashes ("/") as defined in [RFC8610] Appendix G.6.

3.2. YANG Schema Item iDentifier (SID)

Some of the items defined in YANG [RFC7950] require the use of a unique identifier. In both NETCONF [RFC6241] and RESTCONF [RFC8040], these identifiers are implemented using strings. To allow the implementation of data models defined in YANG in constrained devices and constrained networks, a more compact method to identify YANG items is required. This compact identifier, called YANG Schema Item iDentifier (SID), is an unsigned integer. The following items are identified using SIDs:

- identities
- data nodes
- RPCs and associated input(s) and output(s)
- actions and associated input(s) and output(s)
- notifications and associated information
- YANG modules, submodules and features

To minimize their size, SIDs used as keys in inner CBOR maps are typically encoded using deltas. Conversion from SIDs to deltas and back to SIDs are stateless processes solely based on the data serialized or deserialized. These SIDs may also be encoded as absolute number when enclosed by CBOR tag 47.

Mechanisms and processes used to assign SIDs to YANG items and to guarantee their uniqueness are outside the scope of the present specification. If SIDs are to be used, the present specification is used in conjunction with a specification defining this management. One example for such a specification is [I-D.ietf-core-sid].

3.3. Name

This specification also supports the encoding of YANG item identifiers as string, similar as those used by the JSON Encoding of Data Modeled with YANG [RFC7951]. This approach can be used to avoid the management overhead associated to SIDs allocation. The main drawback is the significant increase in size of the encoded data.

YANG item identifiers implemented using names MUST be in one of the following forms:

- 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 data node and its parent node are different. In all other cases, the simple form of the name SHOULD be used.

Definition example:

```
module example-foomod {
  container top {
    leaf foo {
      type uint8;
    }
  }
}

module example-barmod {
  import example-foomod {
    prefix "foomod";
  }
  augment "/foomod:top" {
    leaf bar {
      type boolean;
    }
  }
}
```

A valid CBOR encoding of the 'top' container is as follow.

CBOR diagnostic notation:

```
{
  "example-foomod:top": {
    "foo": 54,
    "example-barmod:bar": true
  }
}
```

Both the 'top' container and the 'bar' leaf defined in a different YANG module as its parent container are encoded as namespace qualified names. The 'foo' leaf defined in the same YANG module as its parent container is encoded as simple name.

4. Encoding of YANG Schema Node Instances

Schema node instances defined using the YANG modeling language are encoded using CBOR [RFC7049] based on the rules defined in this section. We assume that the reader is already familiar with both YANG [RFC7950] and CBOR [RFC7049].

4.1. The 'leaf'

A 'leaf' MUST be encoded accordingly to its datatype using one of the encoding rules specified in Section 6.

The following examples shows the encoding of a 'hostname' leaf using a SID or a name.

Definition example from [RFC7317]:

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

4.1.1. Using SIDs in keys

CBOR diagnostic notation:

```
{
  1752 : "myhost.example.com"     / hostname (SID 1752) /
}
```

CBOR encoding:

```
A1                                          # map(1)
   19 06D8                                  # unsigned(1752)
   72                                       # text(18)
      6D79686F73742E6578616D706C652E636F6D # "myhost.example.com"
```

4.1.2. Using names in keys

CBOR diagnostic notation:

```
{
  "ietf-system:hostname" : "myhost.example.com"
}
```

CBOR encoding:

```
A1                                          # map(1)
   74                                       # text(20)
      696574662D73797374656D3A686F73746E616D65
   72                                       # text(18)
      6D79686F73742E6578616D706C652E636F6D
```

4.2. The 'container' and other collections

Collections such as containers, list instances, notification contents, rpc inputs, rpc outputs, action inputs and action outputs MUST be encoded using a CBOR map data item (major type 5). A map is comprised of pairs of data items, with each data item consisting of a key and a value. Each key within the CBOR map is set to a schema node identifier, each value is set to the value of this schema node instance according to the instance datatype.

This specification supports two type of CBOR keys; SID as defined in Section 3.2 and names as defined in Section 3.3.

The following examples shows the encoding of a 'system-state' container instance using SIDs or names.

Definition example from [RFC7317]:

```
typedef date-and-time {
  type string {
    pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|[\+\-]
             \d{2}:\d{2})';
  }
}

container system-state {

  container clock {
    leaf current-datetime {
      type date-and-time;
    }

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

4.2.1. Using SIDs in keys

In the context of containers and other collections, CBOR map keys within inner CBOR maps can be encoded using deltas or SIDs. In the case of deltas, they MUST be encoded using a CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual delta value. In the case of SID, they are encoded using the SID value enclosed by CBOR tag 47 as defined in Section 8.1.

Delta values are computed as follows:

- In the case of a 'container', deltas are equal to the SID of the current schema node minus the SID of the parent 'container'.
- In the case of a 'list', deltas are equal to the SID of the current schema node minus the SID of the parent 'list'.
- In the case of an 'rpc input' or 'rcp output', deltas are equal to the SID of the current schema node minus the SID of the 'rpc'.
- In the case of an 'action input' or 'action output', deltas are equal to the SID of the current schema node minus the SID of the 'action'.
- In the case of an 'notification content', deltas are equal to the SID of the current schema node minus the SID of the 'notification'.

CBOR diagnostic notation:

```
{
  1720 : {                              / system-state (SID 1720) /
    1 : {                               / clock  (SID 1721) /
      2 : "2015-10-02T14:47:24Z-05:00", / current-datetime(SID 1723)/
      1 : "2015-09-15T09:12:58Z-05:00"  / boot-datetime (SID 1722) /
    }
  }
}
```

CBOR encoding:

```
A1                                      # map(1)
   19 06B8                              # unsigned(1720)
   A1                                   # map(1)
      01                                # unsigned(1)
      A2                                # map(2)
         02                             # unsigned(2)
         78 1A                          # text(26)
            323031352D31302D30325431343A34373A32345A2D30353A3030
         01                             # unsigned(1)
         78 1A                          # text(26)
            323031352D30392D31355430393A31323A35385A2D30353A3030
```

Figure 2: System state clock encoding

4.2.2. Using names in keys

CBOR map keys implemented using names MUST be encoded using a CBOR text string data item (major type 3). A namespace-qualified name MUST be used each time the namespace of a schema node and its parent differ. In all other cases, the simple form of the name MUST be used. Names and namespaces are defined in [RFC7951] section 4.

The following example shows the encoding of a 'system' container instance using names.

Definition example from [RFC7317]:

```
typedef date-and-time {
  type string {
    pattern '\d{4}-\d{2}-\d{2}T\d{2}:\d{2}:\d{2}(\.\d+)?(Z|[\+\-]
             \d{2}:\d{2})';
  }
}

container system-state {

  container clock {
    leaf current-datetime {
      type date-and-time;
    }

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

CBOR diagnostic notation:

```
{
  "ietf-system:system-state" : {
    "clock" : {
      "current-datetime" : "2015-10-02T14:47:24Z-05:00",
      "boot-datetime" : "2015-09-15T09:12:58Z-05:00"
    }
  }
}
```

CBOR encoding:

```
A1                                      # map(1)
   78 18                                # text(24)
   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 instance containing two entries, "ietf.org" and "ieee.org".

Definition example [RFC7317]:

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

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

4.3.1. Using SIDs in keys

CBOR diagnostic notation:

```
{
  1746 : [ "ietf.org", "ieee.org" ]     / search (SID 1746) /
}
```

CBOR encoding:

```
A1                      # map(1)
   19 06D2              # unsigned(1746)
   82                   # array(2)
      68                # text(8)
         696574662E6F7267 # "ietf.org"
      68                # text(8)
         696565652E6F7267 # "ieee.org"
```

4.3.2. Using names in keys

CBOR diagnostic notation:

```
{
  "ietf-system:search" : [ "ietf.org", "ieee.org" ]
}
```

CBOR encoding:

```
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' instance(s)

A list or a subset of a list MUST be encoded using a CBOR array data item (major type 4). Each list instance within this CBOR array is encoded using a CBOR map data item (major type 5) based on the encoding rules of a collection as defined in Section 4.2.

It is important to note that this encoding rule also apply to a single 'list' instance.

The following examples show the encoding of a 'server' list using SIDs or names.

Definition example from [RFC7317]:

```
   list server {
     key name;

     leaf name {
       type string;
     }
     choice transport {
       case udp {
         container udp {
           leaf address {
             type host;
             mandatory true;
           }
           leaf port {
             type port-number;
           }
         }
       }
     }
     leaf association-type {
       type enumeration {
         enum server;
         enum peer;
         enum pool;
       }
       default server;
     }
     leaf iburst {
       type boolean;
       default false;
     }
     leaf prefer {
       type boolean;
       default false;
     }
   }
```

4.4.1. Using SIDs in keys

The encoding rules of each 'list' instance are defined in Section 4.2.1. Deltas of list members are equal to the SID of the current schema node minus the SID of the 'list'.

CBOR diagnostic notation:

```
{
  1756 : [                      / server (SID 1756) /
    {
      3 : "NRC TIC server",     / name (SID 1759) /
      5 : {                     / udp (SID 1761) /
        1 : "tic.nrc.ca",       / address (SID 1762) /
        2 : 123                 / port (SID 1763) /
      },
      1 : 0,                    / association-type (SID 1757) /
      2 : false,                / iburst (SID 1758) /
      4 : true                  / prefer (SID 1760) /
    },
    {
      3 : "NRC TAC server",     / name (SID 1759) /
      5 : {                     / udp (SID 1761) /
        1 : "tac.nrc.ca"        / address (SID 1762) /
      }
    }
  ]
}
```

CBOR encoding:

```
A1                                      # map(1)
   19 06DC                              # unsigned(1756)
   82                                   # array(2)
      A5                                # map(5)
         03                             # unsigned(3)
         6E                             # text(14)
            4E524320544943207365726572 # "NRC TIC server"
         05                             # unsigned(5)
         A2                             # map(2)
            01                          # unsigned(1)
            6A                          # text(10)
               7469632E6E72632E6361     # "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)
            4E524320544143207365726572 # "NRC TAC server"
         05                             # unsigned(5)
         A1                             # map(1)
            01                          # unsigned(1)
            6A                          # text(10)
               7461632E6E72632E6361     # "tac.nrc.ca"
```

4.4.2. Using names in keys

The encoding rules of each 'list' instance are defined in Section 4.2.2.

CBOR diagnostic notation:

```
{
  "ietf-system:server" : [
    {
      "name" : "NRC TIC server",
      "udp" : {
        "address" : "tic.nrc.ca",
        "port" : 123
      },
      "association-type" : 0,
      "iburst" : false,
      "prefer" : true
    },
    {
      "name" : "NRC TAC server",
      "udp" : {
        "address" : "tac.nrc.ca"
      }
    }
  ]
}
```

CBOR encoding:

```
A1                                      # map(1)
   72                                   # text(18)
      696574662D73797374656D3A736572766572
   82                                   # array(2)
      A5                                # map(5)
         64                             # text(4)
            6E616D65                    # "name"
         6E                             # text(14)
            4E5243205449432073657276657
         63                             # text(3)
            756470                      # "udp"
         A2                             # map(2)
            67                          # text(7)
               61646472657373           # "address"
            6A                          # text(10)
               7469632E6E72632E6361     # "tic.nrc.ca"
            64                          # text(4)
               706F7274                 # "port"
            18 7B                       # unsigned(123)
         70                             # text(16)
            6173736F63696174696F6E2D74797065
         00                             # unsigned(0)
         66                             # text(6)
            696275727374                # "iburst"
         F4                             # primitive(20)
         66                             # text(6)
            707265666572                # "prefer"
         F5                             # primitive(21)
      A2                                # map(2)
         64                             # text(4)
            6E616D65                    # "name"
         6E                             # text(14)
            4E5243205441432073657276657
         63                             # text(3)
            756470                      # "udp"
         A1                             # map(1)
            67                          # text(7)
               61646472657373           # "address"
            6A                          # text(10)
               7461632E6E72632E6361     # "tac.nrc.ca"
```

4.5. The 'anydata'

An anydata serves as a container for an arbitrary set of schema nodes that otherwise appear as normal YANG-modeled data. An anydata instance is encoded using the same rules as a container, i.e., CBOR map. The requirement that anydata content can be modeled by YANG implies the following:

- CBOR map keys of any inner schema nodes MUST be set to valid deltas or names.

- The CBOR array MUST contain either unique scalar values (as a leaf-list, see Section 4.3), or maps (as a list, see Section 4.4).

- CBOR map values MUST follow the encoding rules of one of the datatypes listed in Section 4.

The following example shows a possible use of an anydata. In this example, an anydata is used to define a schema node containing a notification event, this schema node can be part of a YANG list to create an event logger.

Definition example:

```
module event-log {
  ...
  anydata last-event;                # SID 60123
```

This example also assumes the assistance of the following notification.

```
module example-port {
  ...

  notification example-port-fault {  # SID 60200
    leaf port-name {                 # SID 60201
      type string;
    }
    leaf port-fault {                # SID 60202
      type string;
    }
  }
}
```

4.5.1. Using SIDs in keys

CBOR diagnostic notation:

```
{
  60123 : {                   / last-event (SID 60123) /
    77 : {                    / event (SID 60200) /
      1 : "0/4/21",           / port-name (SID 60201) /
      2 : "Open pin 2"        / port-fault (SID 60202) /
    }
  }
}
```

CBOR encoding:

```
A1                              # map(1)
   19 EADB                      # unsigned(60123)
   A1                           # map(1)
      18 4D                     # unsigned(77)
      A2                        # map(2)
         18 4E                  # unsigned(78)
         66                     # text(6)
            302F342F3231        # "0/4/21"
         18 4F                  # unsigned(79)
         6A                     # text(10)
            4F70656E2070696E2032 # "Open pin 2"
```

In some implementations, it might be simpler to use the absolute SID tag encoding for the anydata root element. The resulting encoding is as follow:

```
{
  60123 : {                   / last-event (SID 60123) /
    47(60200) : {             / event (SID 60123) /
      1 : "0/4/21",           / port-name (SID 60201) /
      2 : "Open pin 2"        / port-fault (SID 60202) /
    }
  }
}
```

4.5.2. Using names in keys

CBOR diagnostic notation:

```
{
  "event-log:last-event" : {
    "example-port: example-port-fault" : {
      "port-name" : "0/4/21",
      "port-fault" : "Open pin 2"
    }
  }
}
```

CBOR encoding:

```
A1                                      # map(1)
   74                                   # text(20)
      6576656E742D6C6F673A6C6173742D6576656E74
   A1                                   # map(1)
      78 20                             # text(32)
         6578616D706C652D706F72743A206578616D7
         06C652D706F72742D6661756C74
      A2                                # map(2)
         69                             # text(9)
            706F72742D6E616D65          # "port-name"
         66                             # text(6)
            302F342F3231                # "0/4/21"
         6A                             # text(10)
            706F72742D6661756C74        # "port-fault"
         6A                             # text(10)
            4F70656E2070696E2032        # "Open pin 2"
```

4.6. The 'anyxml'

An anyxml schema node is used to serialize an arbitrary CBOR content, i.e., its value can be any CBOR binary object. anyxml value MAY contain CBOR data items tagged with one of the tag listed in Section 8.1, these tags shall be supported.

The following example shows a valid CBOR encoded instance consisting of a CBOR array containing the CBOR simple values 'true', 'null' and 'true'.

Definition example from [RFC7951]:

```
module bar-module {
  ...
  anyxml bar;
```

4.6.1. Using SIDs in keys

CBOR diagnostic notation:

```
{
  60000 : [true, null, true]   / bar (SID 60000) /
}
```

CBOR encoding:

```
A1          # map(1)
   19 EA60  # unsigned(60000)
   83       # array(3)
      F5    # primitive(21)
      F6    # primitive(22)
      F5    # primitive(21)
```

4.6.2. Using names in keys

CBOR diagnostic notation:

```
{
  "bar-module:bar" : [true, null, true]   / bar (SID 60000) /
}
```

CBOR encoding:

```
A1                                  # map(1)
   6E                               # text(14)
      6261722D6D6F64756C653A626172 # "bar-module:bar"
   83                               # array(3)
      F5                            # primitive(21)
      F6                            # primitive(22)
      F5                            # primitive(21)
```

5. Encoding of YANG data templates

YANG data templates are data structures defined in YANG but not intended to be implemented as part of a datastore. YANG data templates are defined using the 'yang-data' extension as described by [RFC8040].

YANG data templates MUST be encoded using the encoding rules of a collection as defined in Section 4.2.

Just like YANG containers, YANG data templates can be encoded using either SIDs or names.

Definition example from [I-D.ietf-core-comi]:

```
import ietf-restconf {
  prefix rc;
}

rc:yang-data yang-errors {
  container error {
    leaf error-tag {
      type identityref {
        base error-tag;
      }
    }
    leaf error-app-tag {
      type identityref {
        base error-app-tag;
      }
    }
    leaf error-data-node {
      type instance-identifier;
    }
    leaf error-message {
      type string;
    }
  }
}
```

5.1. Using SIDs in keys

YANG template encoded using SIDs are carried in a CBOR map containing a single item pair. The key of this item is set to the SID assigned to the YANG template container, the value is set the CBOR encoding of this container as defined in Section 4.2.

This example shows a serialization example of the yang-errors template as defined in [I-D.ietf-core-comi] using SIDs as defined in Section 3.2.

CBOR diagnostic notation:

```
{
  1024 : {                        / error  (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 exceeded"        / error-message (SID 1027) /
      }
}
```

CBOR encoding:

```
A1                                      # map(1)
   19 0400                              # unsigned(1024)
   A4                                   # map(4)
      04                                # unsigned(4)
      19 03F3                           # unsigned(1011)
      01                                # unsigned(1)
      19 03FA                           # unsigned(1018)
      02                                # unsigned(2)
      19 06CC                           # unsigned(1740)
      03                                # unsigned(3)
      70                                # text(16)
         4D6178696D756D206578636565646564
```

5.2. Using names in keys

YANG template encoded using names are carried in a CBOR map containing a single item pair. The key of this item is set to the namespace qualified name of the YANG template container, the value is set the CBOR encoding of this container as defined in Section 3.3.

This example shows a serialization example of the yang-errors template as defined in [I-D.ietf-core-comi] using names as defined Section 3.3.

CBOR diagnostic notation:

```
{
  "ietf-comi:error" : {
    "error-tag" : "invalid-value",
    "error-app-tag" : "not-in-range",
    "error-data-node" : "timezone-utc-offset",
    "error-message" : "Maximum exceeded"
  }
}
```

CBOR encoding:

```
A1                                      # map(1)
   6F                                   # text(15)
      696574662D636F6D693A6572726F72     # "ietf-comi:error"
   A4                                   # map(4)
      69                                # text(9)
         6572726F722D746167              # "error-tag"
      6D                                # text(13)
         696E76616C69642D76616C7565      # "invalid-value"
      6D                                # text(13)
         6572726F722D6170702D746167      # "error-app-tag"
      6C                                # text(12)
         6E6F742D696E2D72616E6765        # "not-in-range"
      6F                                # text(15)
         6572726F722D646174612D6E6F6465 # "error-data-node"
      73                                # text(19)
         74696D657A6F6E652D7574632D6F6666736574
                                        # "timezone-utc-offset"
      6D                                # text(13)
         6572726F722D6D657373616765      # "error-message"
      70                                # text(16)
         4D6178696D756D206578636565646564
```

6. Representing YANG Data Types in CBOR

The CBOR encoding of an instance of a leaf or leaf-list schema node depends on the built-in type of that schema node. The following sub-section defined the CBOR encoding of each built-in type supported by YANG as listed in [RFC7950] section 4.2.4. Each subsection shows an example value assigned to a schema node instance of the discussed built-in type.

6.1. The unsigned integer Types

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

The following example shows the encoding of a 'mtu' leaf instance set to 1280 bytes.

Definition example from [RFC8344]:

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

CBOR diagnostic notation: 1280

CBOR encoding: 19 0500

6.2. The integer Types

Leafs of type int8, int16, int32 and int64 MUST be encoded using either CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value.

The following example shows the encoding of a 'timezone-utc-offset' leaf instance set to -300 minutes.

Definition example from [RFC7317]:

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

CBOR diagnostic notation: -300

CBOR encoding: 39 012B

6.3. The 'decimal64' Type

Leafs of type decimal64 MUST be encoded using a decimal fraction as defined in [RFC7049] section 2.4.3.

The following example shows the encoding of a 'my-decimal' leaf instance set to 2.57.

Definition example from [RFC7317]:

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

CBOR diagnostic notation: 4([-2, 257])

CBOR encoding: C4 82 21 19 0101

6.4. The 'string' Type

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

The following example shows the encoding of a 'name' leaf instance set to "eth0".

Definition example from [RFC8343]:

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

CBOR diagnostic notation: "eth0"

CBOR encoding: 64 65746830

6.5. The 'boolean' Type

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

The following example shows the encoding of an 'enabled' leaf instance set to 'true'.

Definition example from [RFC7317]:

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

CBOR diagnostic notation: true

CBOR encoding: F5

6.6. The 'enumeration' Type

Leafs of type enumeration MUST be encoded using a CBOR unsigned integer (major type 0) or CBOR negative integer (major type 1), depending on the actual value. Enumeration values are either explicitly assigned using the YANG statement 'value' or automatically assigned based on the algorithm defined in [RFC7950] section 9.6.4.2.

The following example shows the encoding of an 'oper-status' leaf instance set to 'testing'.

Definition example from [RFC7317]:

```
leaf oper-status {
  type enumeration {
    enum up { value 1; }
    enum down { value 2; }
    enum testing { value 3; }
    enum unknown { value 4; }
    enum dormant { value 5; }
    enum not-present { value 6; }
    enum lower-layer-down { value 7; }
  }
}
```

CBOR diagnostic notation: 3

CBOR encoding: 03

To avoid overlap of 'value' defined in different 'enumeration' statements, 'enumeration' defined in a Leafs of type 'union' MUST be encoded using a CBOR text string data item (major type 3) and MUST contain one of the names assigned by 'enum' statements in YANG. The encoding MUST be enclosed by the enumeration CBOR tag as specified in Section 8.1.

Definition example from [RFC7950]:

```
type union {
  type int32;
  type enumeration {
    enum "unbounded";
  }
}
```

CBOR diagnostic notation: 44("unbounded")

CBOR encoding: D8 2C 69 756E626F756E646564

6.7. The 'bits' Type

Leafs of type bits MUST be encoded using a CBOR byte string data item (major type 2). Bits position are either explicitly assigned using the YANG statement 'position' or automatically assigned based on the algorithm defined in [RFC7950] section 9.7.4.2.

Bits position 0 to 7 are assigned to the first byte within the byte string, bits 8 to 15 to the second byte, and subsequent bytes are

assigned similarly. Within each byte, bits are assigned from least to most significant.

The following example shows the encoding of an 'alarm-state' leaf instance with the 'under-repair' and 'critical' flags set.

Definition example from [RFC8348]:

```
typedef alarm-state {
  type bits {
    bit unknown;
    bit under-repair;
    bit critical;
    bit major;
    bit minor;
    bit warning;
    bit indeterminate;
  }
}

leaf alarm-state {
  type alarm-state;
}
```

CBOR diagnostic notation: h'06'

CBOR encoding: 41 06

To avoid overlap of 'bit' defined in different 'bits' statements, 'bits' defined in a Leafs of type 'union' MUST be encoded using a CBOR text string data item (major type 3) and MUST contain a space-separated sequence of names of 'bit' that are set. The encoding MUST be enclosed by the bits CBOR tag as specified in Section 8.1.

The following example shows the encoding of an 'alarm-state' leaf instance defined using a union type with the 'under-repair' and 'critical' flags set.

Definition example:

```
leaf alarm-state-2 {
  type union {
    type alarm-state;
    type bits {
      bit extra-flag;
    }
  }
}
```

CBOR diagnostic notation: 43("under-repair critical")

CBOR encoding: D8 2B 75 756E6465722D726570616972206372697469636616C

6.8. The 'binary' Type

Leafs of type binary MUST be encoded using a CBOR byte string data item (major type 2).

The following example shows the encoding of an 'aes128-key' leaf instance set to 0x1f1ce6a3f42660d888d92a4d8030476e.

Definition example:

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

CBOR diagnostic notation: h'1F1CE6A3F42660D888D92A4D8030476E'

CBOR encoding: 50 1F1CE6A3F42660D888D92A4D8030476E

6.9. The 'leafref' Type

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

The following example shows the encoding of an 'interface-state-ref' leaf instance set to "eth1".

Definition example from [RFC8343]:

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

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

CBOR diagnostic notation: "eth1"

CBOR encoding: 64 65746831

6.10. The 'identityref' Type

This specification supports two approaches for encoding identityref, a YANG Schema Item iDentifier (SID) as defined in Section 3.2 or a name as defined in [RFC7951] section 6.8.

6.10.1. SIDs as identityref

When schema nodes of type identityref are implemented using SIDs, they MUST be encoded using a CBOR unsigned integer data item (major type 0). (Note that no delta mechanism is employed for SIDs as identityref.)

The following example shows the encoding of a 'type' leaf instance set to the value 'iana-if-type:ethernetCsmacd' (SID 1880).

Definition example from [RFC7317]:

```
identity interface-type {
}

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

identity ethernetCsmacd {
  base iana-interface-type;
}

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

CBOR diagnostic notation: 1880

CBOR encoding: 19 0758

6.10.2. Name as identityref

Alternatively, an identityref MAY be encoded using a name as defined in Section 3.3. When names are used, identityref MUST be encoded using a CBOR text string data item (major type 3). If the identity is defined in different module than the leaf node containing the identityref data node, the namespace qualified form MUST be used. Otherwise, both the simple and namespace qualified forms are permitted. Names and namespaces are defined in Section 3.3.

The following example shows the encoding of the identity 'iana-if-type:ethernetCsmacd' using its namespace qualified name. This example is described in Section 6.10.1.

CBOR diagnostic notation: "iana-if-type:ethernetCsmacd"

CBOR encoding: 78 1b
69616E612D69662D747970653A65746865726E657443736D616364

6.11. The 'empty' Type

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

The following example shows the encoding of a 'is-router' leaf instance when present.

Definition example from [RFC8344]:

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

CBOR diagnostic notation: null

CBOR encoding: F6

6.12. The 'union' Type

Leafs of type union MUST be encoded using the rules associated with one of the types listed. When used in a union, the following YANG datatypes are enclosed by a CBOR tag to avoid confusion between different YANG datatypes encoded using the same CBOR major type.

- bits
- enumeration
- identityref
- instance-identifier

See Section 8.1 for the assigned value of these CBOR tags.

As mentioned in Section 6.6 and in Section 6.7, 'enumeration' and 'bits' are encoded as CBOR text string data item (major type 3) when defined within a 'union' type.

The following example shows the encoding of an 'ip-address' leaf instance when set to "2001:db8:a0b:12f0::1".

Definition example from [RFC7317]:

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

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

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

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

CBOR diagnostic notation: "2001:db8:a0b:12f0::1"

CBOR encoding: 74 323030313A6462383A6130623A313266303A3A31

6.13. The 'instance-identifier' Type

This specification supports two approaches for encoding an instance-identifier, one based on YANG Schema Item iDentifier (SID) as defined in Section 3.2 and one based on names as defined in Section 3.3.

6.13.1. SIDs as instance-identifier

SIDs uniquely identify a schema node. In the case of a single instance schema node, i.e. a schema node defined at the root of a YANG module or submodule or schema nodes defined within a container, the SID is sufficient to identify this instance.

In the case of a schema node member of a YANG list, a SID is combined with the list key(s) to identify each instance within the YANG list(s).

Single instance schema nodes MUST be encoded using a CBOR unsigned integer data item (major type 0) and set to the targeted schema node SID.

Schema nodes member of a YANG list MUST be encoded using a CBOR array data item (major type 4) containing the following entries:

- 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 top level list, and follow by each of the subordinate list(s).

Examples within this section assume the definition of a schema node of type 'instance-identifier':

Definition example from [RFC7950]:

```
container system {
  ...
  leaf reporting-entity {
    type instance-identifier;
  }

leaf contact { type string; }

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

First example:

The following example shows the encoding of the 'reporting-entity' value referencing data node instance "/system/contact" (SID 1741).

Definition example from [RFC7317]:

```
container system {

  leaf contact {
    type string;
  }

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

CBOR diagnostic notation: 1741

CBOR encoding: 19 06CD

Second example:

The following example shows the encoding of the 'reporting-entity' value referencing list instance "/system/authentication/user/authorized-key/key-data" (SID 1734) for user name "bob" and authorized-key "admin".

Definition example from [RFC7317]:

```
list user {
  key name;

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

  list authorized-key {
    key name;

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

CBOR diagnostic notation: [1734, "bob", "admin"]

CBOR encoding:

```
83               # array(3)
   19 06C6       # unsigned(1734)
   63            # text(3)
      626F62     # "bob"
   65            # text(5)
      61646D696E # "admin"
```

Third example:

The following example shows the encoding of the 'reporting-entity' value referencing the list instance "/system/authentication/user" (SID 1730) corresponding to user name "jack".

CBOR diagnostic notation: [1730, "jack"]

CBOR encoding:

```
82             # array(2)
   19 06C2     # unsigned(1730)
   64          # text(4)
      6A61636B # "jack"
```

6.13.2. Names as instance-identifier

An "instance-identifier" value is encoded as a string that is analogical to the lexical representation in XML encoding; see Section 9.13.2 in [RFC7950]. However, the encoding of namespaces in instance-identifier values follows the rules stated in Section 3.3, namely:

- 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:

"/ietf-system:system/authentication/user[name='bob']/authorized-key [name='admin']/key-data"

CBOR encoding:

```
78 59
   2F696574662D73797374656D3A73797374656D2F61757468656E74696361
   74696F6E2F757365725B6E616D653D27626F62275D2F617574686F72697A
   65642D6B65790D0A5B6E616D653D2761646D696E275D2F6B65792D64617461
```

Third example:

This example is described in Section 6.13.1.

CBOR diagnostic notation:

"/ietf-system:system/authentication/user[name='bob']"

CBOR encoding:

```
78 33
   2F696574662D73797374656D3A73797374656D2F61757468656E74696361
   74696F6E2F757365725B6E616D653D27626F62275D
```

7. Security Considerations

The security considerations of [RFC7049] and [RFC7950] apply.

This document defines an alternative encoding for data modeled in the YANG data modeling language. As such, this encoding does not contribute any new security issues in addition of those identified for the specific protocol or context for which it is used.

To minimize security risks, software on the receiving side SHOULD reject all messages that do not comply to the rules of this document and reply with an appropriate error message to the sender.

8. IANA Considerations

8.1. CBOR Tags Registry

This specification requires the assignment of CBOR tags for the following YANG datatypes. These tags are added to the CBOR Tags Registry as defined in section 7.2 of [RFC7049].

Tag	Data Item	Semantics	Reference
43	byte string	YANG bits datatype ; see Section 6.7.	[this]
44	unsigned integer	YANG enumeration datatype ;see Section 6.6.	[this]
45	unsigned integer or text string	YANG identityref datatype ; see Section 6.10.	[this]
46	unsigned integer or text string or array	YANG instance-identifier datatype; see Section 6.13.	[this]
47	unsigned integer	YANG Schema Item iDentifier ; see Section 3.2.	[this]

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

9. Acknowledgments

This document has been largely inspired by the extensive works done by Andy Bierman and Peter van der Stok on [I-D.ietf-core-comi]. [RFC7951] has also been a critical input to this work. The authors would like to thank the authors and contributors to these two drafts.

The authors would also like to acknowledge the review, feedback, and comments from Ladislav Lhotka and Juergen Schoenwaelder.

10. References

10.1. Normative References

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

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, DOI 10.17487/RFC6241, June 2011, <https://www.rfc-editor.org/info/rfc6241>.

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

[RFC7950] Bjorklund, M., Ed., "The YANG 1.1 Data Modeling Language", RFC 7950, DOI 10.17487/RFC7950, August 2016, <https://www.rfc-editor.org/info/rfc7950>.

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

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

10.2. Informative References

[I-D.ietf-core-comi]
Veillette, M., Stok, P., Pelov, A., Bierman, A., and I. Petrov, "CoAP Management Interface", draft-ietf-core-comi-07 (work in progress), July 2019.

[I-D.ietf-core-sid]
Veillette, M., Pelov, A., and I. Petrov, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-07 (work in progress), July 2019.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <https://www.rfc-editor.org/info/rfc7317>.

[RFC7951] Lhotka, L., "JSON Encoding of Data Modeled with YANG", RFC 7951, DOI 10.17487/RFC7951, August 2016, <https://www.rfc-editor.org/info/rfc7951>.

[RFC8040] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", RFC 8040, DOI 10.17487/RFC8040, January 2017, <https://www.rfc-editor.org/info/rfc8040>.

[RFC8259] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", STD 90, RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[RFC8343] Bjorklund, M., "A YANG Data Model for Interface Management", RFC 8343, DOI 10.17487/RFC8343, March 2018, <https://www.rfc-editor.org/info/rfc8343>.

[RFC8344] Bjorklund, M., "A YANG Data Model for IP Management", RFC 8344, DOI 10.17487/RFC8344, March 2018, <https://www.rfc-editor.org/info/rfc8344>.

[RFC8348] Bierman, A., Bjorklund, M., Dong, J., and D. Romascanu, "A YANG Data Model for Hardware Management", RFC 8348, DOI 10.17487/RFC8348, March 2018, <https://www.rfc-editor.org/info/rfc8348>.

Authors' Addresses

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

Email: michel.veillette@trilliantinc.com

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

Email: ivaylo@ackl.io

Alexander Pelov
Acklio
1137A avenue des Champs Blancs
Cesson-Sevigne, Bretagne 35510
France

Email: a@ackl.io

Network Working Group
Internet-Draft
Intended status: Informational
Expires: May 4, 2017

J. Jimenez
Ericsson

October 31, 2016

CoAP functionality expected in a LWM2M system
draft-jimenez-t2trg-coap-functionality-lwm2m-00

Abstract

This document provides a strawman summary of information that should be used for the LWM2M specification [LWM2M]. LWM2M is based on CoAP, on top of which it describes certain management interfaces and data models that go beyond the CoAP specifications itself. However LWM2M does not describe all behavior that should be expected from implementations of the CoAP specifications. This document attempts to clarify what should be present in a LWM2M system beyond what is specified in the LWM2M documents. Additionally, this document also adds information about IPSO Objects [IPSO] and their usage with LWM2M as application protocol.

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

Table of Contents

1. Introduction

The current LWM2M protocol is probably the main Device Management protocol based on CoAP today. It defines the application layer communication protocol between a LWM2M Server and a LWM2M Client, which is located in a LWM2M Device.

2. Terminology

The LWM2M Specification tends to use its own terminology for client, server, etc. In this document, we use the existing terminology from [RFC7252].

For example the use of LWM2M "Client" and "Server" and the roles they play has confused developers that are initiating on the protocol, mainly because a CoAP server runs on the device, just like a LWM2M client does. Moreover, most LWM2M devices will often work both as client and server depending on the interfaces used, it would be good to explore the use of terms like "servients" for devices that regularly support both.

Similarly, the reference to existing drafts of RFCs often can mislead the reader to believe that the full RFC has been implemented. It would be better to state the support to an IETF CoRE WG document when applicable.

For example, the Registration interface in LWM2M is based on the CoAP Resource Directory. However, it is not sufficient to implement just

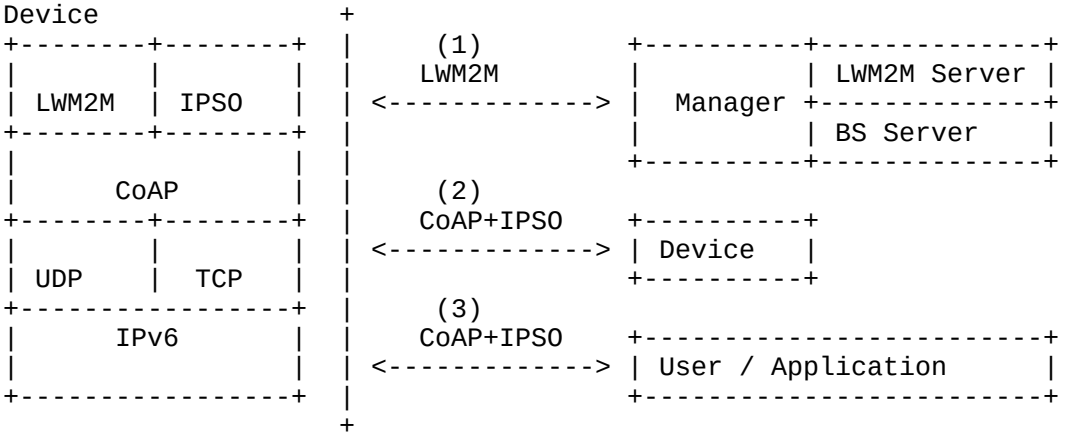
the interface described to obtain the benefits provided by the CoAP Resource Directory.

3. Interaction Model

LWM2M has been created with a strong focus on centralizing control and management. Devices set associations with their manager and all traffic is directed to the cloud. All this is fine but in the process some functionalities that could be used locally device to device and device to application have not been explicitly described.

Below we have common configurations that make use of LWM2M.

- (1) Device and Manager configuration.
- (2) Device to Device configuration.
- (3) Device to Application configuration.



3.1. Device and Manager configuration.

This is covered by common LWM2M compliant implementations we have today. However there are upcoming RFCs and drafts that greatly enhance LWM2M with more CoAP features.

For example TCP support is soon going to be added to CoAP. The draft [I-D.ietf-core-coap-tcp-tls] outlines the changes required to use CoAP over TCP, TLS, and WebSockets transports.

Support for features like PATCH/FETCH [I-D.ietf-core-etch] could be greatly beneficial for things like firmware upgrade or observing relatively large sets of resources.

For systems in which endpoints work behind a gateway or use LWM2M for managing the gateways, it might be good to implement other types of cryptographic protection than DTLS. For example some of the setups using OSCoAP [I-D.ietf-core-object-security] allow for "smarter" gateways.

3.2. Device to Device configuration.

Beyond what is described in the LWM2M documentation, devices will often talk to each other. Specially in cases when all devices are under the same subnet, this could be pretty common. For example devices could be more resilient if they did not have to contact their manager constantly; in case of lack of internet connectivity the local IoT network would still function. Managers could just set policies on the devices and they would operate more autonomously.

For this setup to take place, LWM2M would use more of the device-to-device functionality of CoAP. A more complete Resource Directory implementation [I-D.ietf-core-resource-directory] would be needed, either on the LWM2M server in addition to the registration interface or standalone. Devices should be able to perform lookup on that RD and get the series of links to resources elsewhere. They should be able to find new functionality through /.well-known/core. If not, they should be able to use IP multicast as expressed on [RFC7390].

Needless to say, it is assumed that devices would be running a CoAP Server on them and would support CoAP Observe [RFC7641], so that devices can subscribe to updates from one another, thus becoming more autonomous.

There are also updates on ACE security framework, that allow for securing the communication between two devices via an Authorization Server [I-D.ietf-ace-oauth-authz].

The current LWM2M Data Model needs more expressiveness when it comes to data types; More on that in Section 4. Also Web Linking will be dealt at Section 5.

3.3. Device to Application configuration.

In some other cases applications would be running on the phone connecting locally to sensors and/or control actuators. A smartphone can access directly a CoAP home sensor using a mutually authenticated ’https’ request, provided its home router runs a HTTP to CoAP (HC) proxy and is configured with the appropriate certificate. For this scenario to happen, the GW should implement a HC proxy. It is highly recommended then that they make use of [I-D.ietf-core-http-mapping] to properly do the URI mapping and specific ABNF queries.

Just like other devices, smartphone applications should be able to discover devices using standard methods, thus they would need access to the RD as well.

4. Data Model

The LWM2M Object Model is specified in [LWM2M]. Other models that build on it like IPSOs or OneM2M have spawned out of it. They normally introduce incremental features. They usually allow for performing any set of operations on a device through a CoAP interface. Resources are exposed as Objects using the same data model used for management.

For example, in the case of a temperature sensor we can access and subscribe to the readings of the device (using [IPSO]).

Req: GET /3303/0/5700 Observe_Option=1
Res: 2.05 Content (25 C)
Res (Notify): 2.05 Content (26 C)

There has also been much work on different serialization and compression mechanisms that LWM2M could consider adopting. A serialized JSON file like the one below could be greatly compressed (about 46% max, depending on the case) using CBOR representation format [RFC7049] instead.

```
{       "e": [{
        "bn": "/3303/0",
        "e": [{
            "n": "5700",
            "v": 20.0 }, {
            "n": "5701",
            "v": "c" }, {
            "n": "5603",
            "v": 10 }, {
            "n": "5604",
            "v": 40
        }],    }, {
        "bn": "/3302/0",
        "e": [{
            "n": "5500",
            "v": true }, {
            "n": "5501",
            "v": 23
}       ]}]}
```

LWM2M ResourceIDs at the moment have no specific semantic meaning like ObjectIDs do. Adding a similar registry for ResourceIDs could

be useful. Specially to those using LWM2M for their applications. For example IPSO uses such ResourceIDs to register resources univocally, so that the string _5701_ consistently represents units.

5. Web Linking

One thing that that could be very useful in the future is some form of Web Link resource type. ObjectLinks are not sufficient to represent links between devices or applications. There has been much work on web linking on [RFC6690] that could be used in the LWM2M spec. For example a new Data Type named "Web Link" could be a simple, yet useful addition. Instead of the current _ObjectID:InstanceID_ expressed now, a full WebLink would be used. That would take advantage of other features like [I-D.ietf-core-links-json] or even newer Object Models.

Other use cases contemplate some form of Object Redirection to help decouple management and applications. LWM2M expects that the management servers will observe resources and collect telemetry on the management server itself. If LWM2M is to be used as application protocol as well as management, it should provide a way for applications or CoAP Clients to observe resources on the devices, together with their required credentials. Such credentials should be stored on the device in some way, maybe a new Object.

6. Collaboration

To further develop the relationship between the LWM2M specification and other specifications based on CoAP, it would also be advisable to foster collaboration between organizations developing CoAP-based standard implementations. At the moment there is no forum for inter group communication nor discussion. That should change.

The IETF CoRE WG has quite some people also interested in device management. Communication would be mutually beneficial. Example of that work is on COMI [I-D.ietf-core-yang-cbor] or data model translation.

OMA LWM2M already has benefited from workshops that gather most of the industry, such as [IOTSI] and [IOTSU]. Similarly, specifications can be developed in the IETF with a view to be directly usable in LWM2M.

7. 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)", draft-ietf-ace-oauth-authz-04 (work in progress), October 2016.

[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.ietf-core-etch]
Stok, P., Bormann, C., and A. Sehgal, "Patch and Fetch Methods for Constrained Application Protocol (CoAP)", draft-ietf-core-etch-03 (work in progress), October 2016.

[I-D.ietf-core-http-mapping]
Castellani, A., Loreto, S., Rahman, A., Fossati, T., and E. Dijk, "Guidelines for HTTP-to-CoAP Mapping Implementations", draft-ietf-core-http-mapping-16 (work in progress), October 2016.

[I-D.ietf-core-links-json]
Li, K., Rahman, A., and C. Bormann, "Representing CoRE Formats in JSON and CBOR", draft-ietf-core-links-json-06 (work in progress), July 2016.

[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.ietf-core-resource-directory]
Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", draft-ietf-core-resource-directory-08 (work in progress), July 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.

[IOTSI] IAB, "IoT Workshop for Semantic Interoperability (IOTSI)", 2016, <https://www.iab.org/activities/workshops/iotsi/>.

[IOTSU] IAB, "Internet of Things Software Update Workshop (IoTSU)", 2016, <https://www.iab.org/activities/workshops/iotsu/>.

[IPSO] IPSO, "IPSO Object Model", n.d., <http://ipso-alliance.github.io/pub/>.

[LWM2M] OMA, "LWM2M specification", n.d., <http://technical.openmobilealliance.org/Technical/technical-information/release-program/current-releases/oma-lightweightm2m-v1-0>.

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

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

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <http://www.rfc-editor.org/info/rfc7390>.

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

Author’s Address

Jaime Jimenez
Ericsson

Phone: +358-442-992-827
Email: jaime.jimenez@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.

Table of Contents

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 folling 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 constained 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 successfull 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.

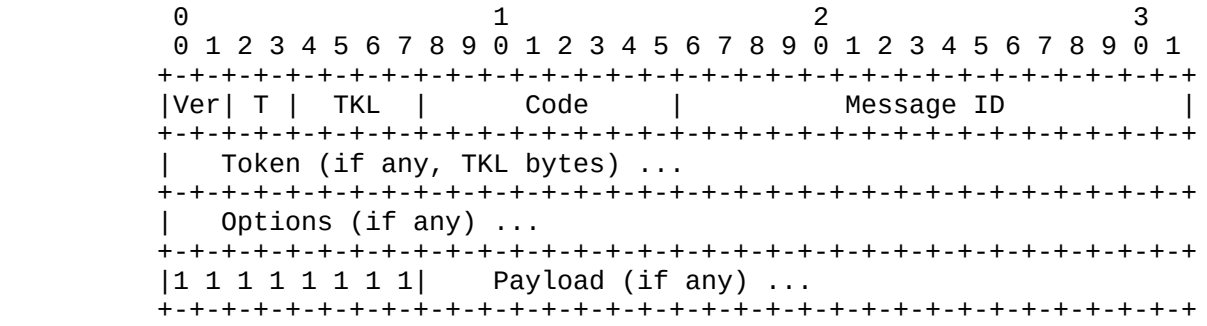
5. Message Bindings

This section describes BRSKI to CoAP message mappings.

CoAP defines confirmed (CON), acknowledgements (ACK), reset (RST) and non-corfirmed (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.

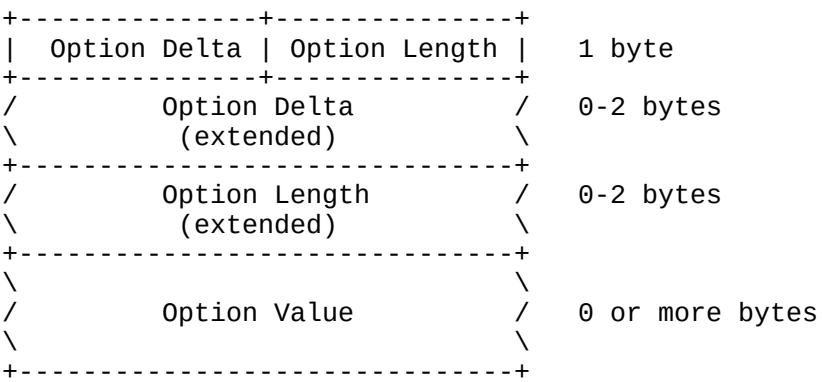
The HTTP responses in BRSKI are translated to COAP Response Codes as explained in [RFC7252] section 5.3.1

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?)

MIIMOQYJKoZIhvcNAQcCoIIMKjCCDCYCAQExADALBgkqhkiG9w0BBwGgggwMMIIC
+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 \
           DALBgkqhkiG9w0BBwGgggwMMIIC...
```

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. requestauditttoken, 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
DwAwggEKAoIBAQClNp+kdz+Nj8XpEp9kaumWxDZ3eFYJpQKz9ddD5e5OzUeCm103
ZIXQIxc0eVtMCatnRr3dnZRCAxGjwbqoB3eKt29/XSQffVv+odbyw0WdkQOIbntC
Qry8YdcBZ+8LjI/N7M2krmjmoSLmLwU2V4aNKf0YMLR5Krmah3Ik31jmYCSvwTnv
6mx6pr2pTJ82JavhTEIIt/fAYq1RYhkM1CXoBL+yhEoDanN7TzC94skfS3VV+f53
J9SkUxTYcy1Rw0k3VXfxWwy+cSKEPREl7I6k0YeKtDEVAgBIEYM/L1S69RXTLuji
rwnqSRjOquzkAkD31BE961KZCxeYGrhxaR4PAgMBAAGjUjBQMA4GA1UdDwEB/wQE
AwIEsDAdBgNVHQ4EFgQU/qDdB6ii6icQ8wGMXvy1jfE4xtUwHwYDVR0jBBgwFoAU
scRp5lujBKfYl6OLO7+5arIyQjwwDQYJKoZIhvcNAQEFBQADggEBACmxg1hvL6+7
a+lFTARoxainBx5gxdZ9omSb0L+qL+4PDvg/+KHzKsDnMCrcU6M4YP5n0EDKmGa6
4lY8fbET4tt7juJg6ixb95/760Th0vuctwkGr6+D6ETTfqyHnrbhX3lAhnB+0Ja7
o1gv4CWxh1I8aRaTXdpOHORvN0SMXdcrlCys2vrtOl+LjR2a3kajJO6eQ5leOdzF
QlZfOPhaLWen0e2BLNJI0vsC2Fa+2LMCnfC38XfGALa5A8e7fNHXWZBjXZLBCza3
rEs9Mlh2CjA/ocSC/WxmMvd+Eqnt/FpggRy+F8IZSRvBaRUCtGE1lgDmu6AFUxce
R4POrT2xz8ChADEA

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 = ...
```

[[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".]]

[[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 then 2s, but in this case they should send an CoAP ACK every 2s while processing.]]

[[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.]]

-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.

[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.

Table of Contents

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:

- 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"

- 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:

- 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

- A means for CoAP clients to interact with a CoRE resource directory interface for requesting and discovering alternative transports and locations of CoAP resources

- New Resource Directory parameter types enabling the above-mentioned features.

- 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.

- 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.

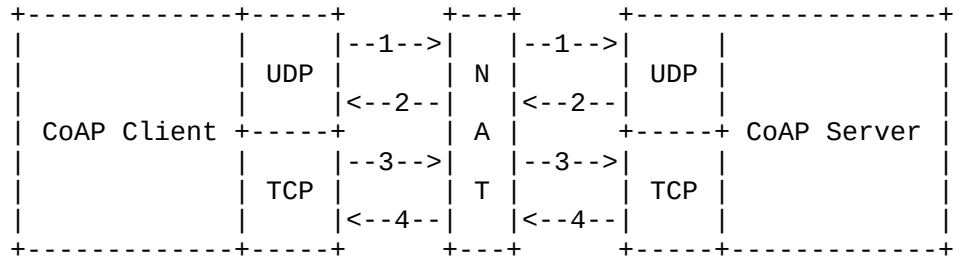


Figure 1: CoAP Client initially accesses CoAP Server over UDP and then switching to TCP

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.

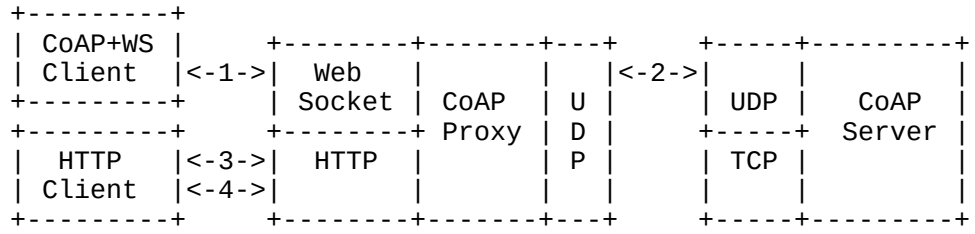


Figure 2: Proxying and returning a resource's alternate cached representations to multiple clients

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 comunication), 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.

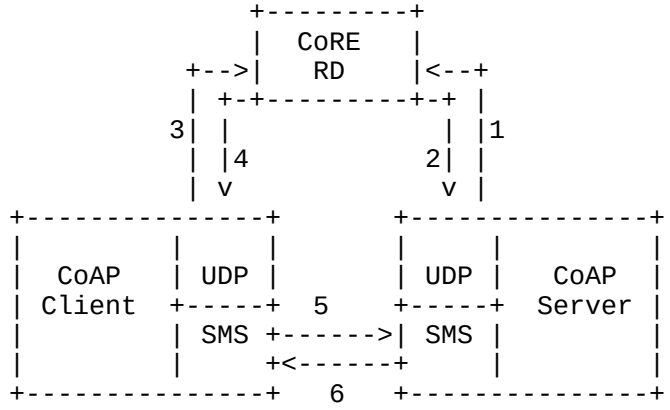


Figure 3: CoAP client interacting with RD to discover a server's SMS-based endpoint

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.

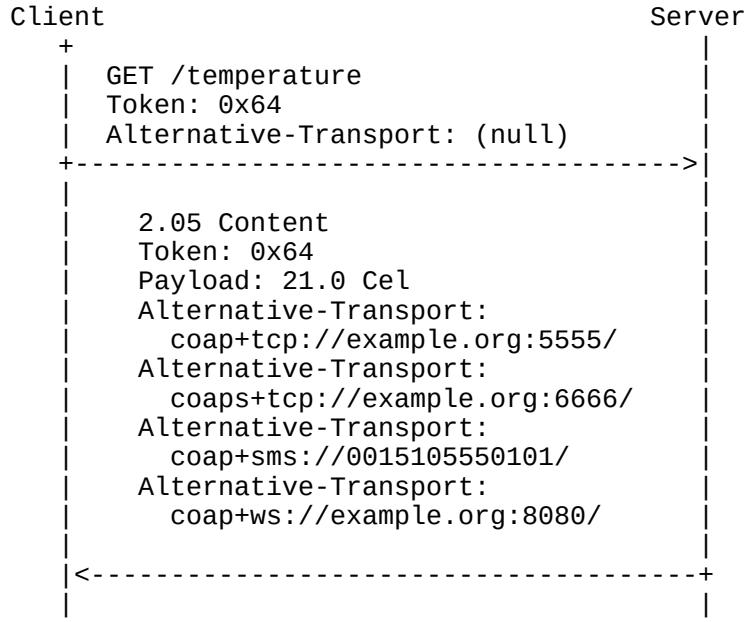


Figure 5: Requesting all available alternative transports on the server, and their locations

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.

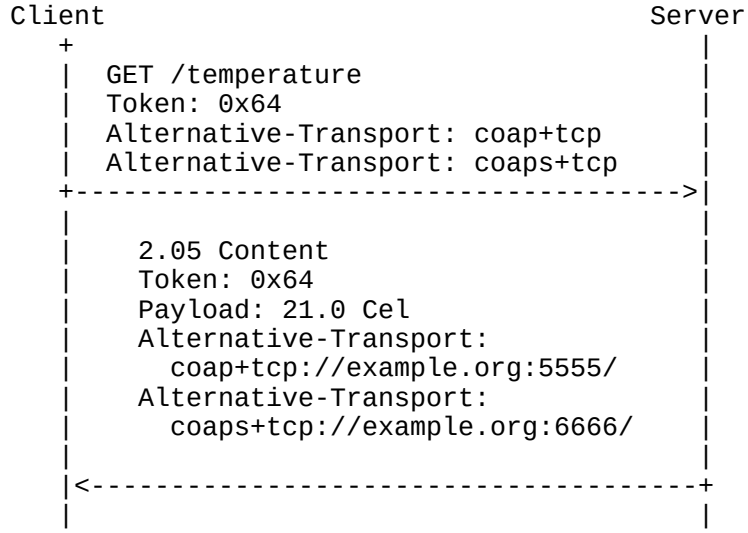


Figure 6: Requesting TCP-based alternative transports on the server, and their locations

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.

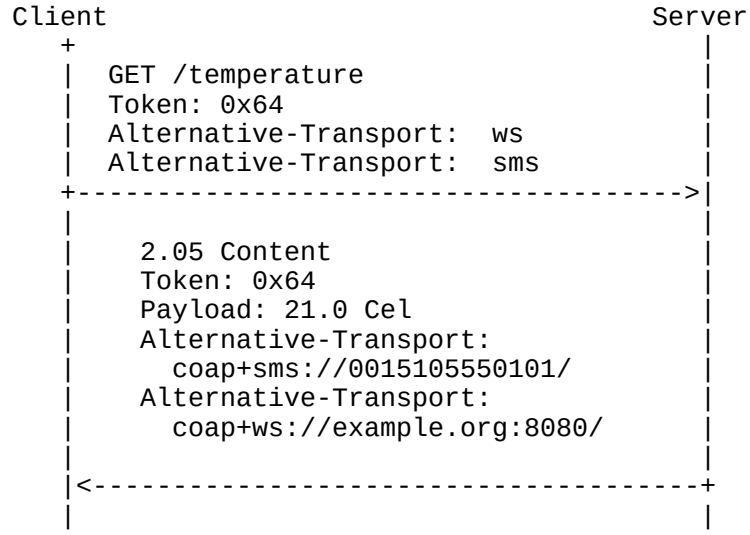


Figure 7: Requesting WebSocket- and SMS-based alternative transports on the server, and their locations

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 negotations 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-transports]
Silverajan, B. and T. Savolainen, "CoAP Communication with Alternative Transports", draft-silverajan-core-coap-alternative-transports-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.

Table of Contents

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:

- 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.

- 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 reponses 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.

- 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

Table of Contents

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.

- 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].

- 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.

- 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.

- 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.

- Pure listener: member of a multicast group that is configured as listener and never replies back to multicasters after receiving multicast messages.

- 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.

- 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.

- 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.

- Multicast communication topology: this document considers both 1-to-N (one multicaster and multiple listeners) and M-to-N (multiple multicasters 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.

- 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.

- 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].

- 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.

- 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.

- 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:

- Data replay protection: replayed group request messages or response messages MUST be detected.

- 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.

- 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).

- 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.

- 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 uniquess 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.

- 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.

- 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

- 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
]
```

- 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.

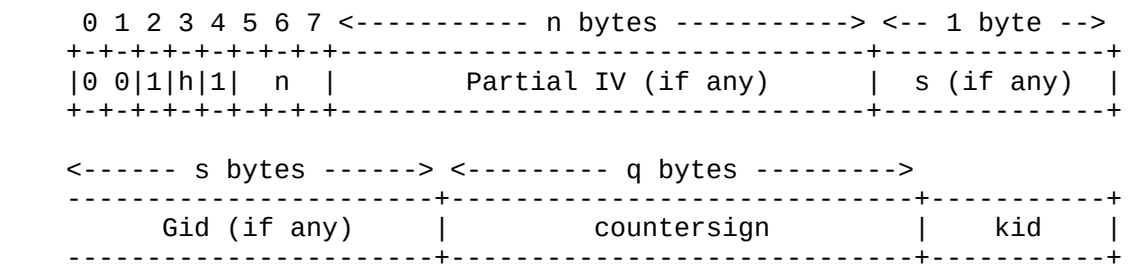


Figure 2: Object-Security Value

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>.

[RFC2094] Harney, H. and C. Muckenhirn, "Group Key Management Protocol (GKMP) Architecture", RFC 2094, DOI 10.17487/RFC2094, July 1997, <https://www.rfc-editor.org/info/rfc2094>.

[RFC2627] Wallner, D., Harder, E., and R. Agee, "Key Management for Multicast: Issues and Architectures", RFC 2627, DOI 10.17487/RFC2627, June 1999, <https://www.rfc-editor.org/info/rfc2627>.

[RFC3376] Cain, B., Deering, S., Kouvelas, I., Fenner, B., and A. Thyagarajan, "Internet Group Management Protocol, Version 3", RFC 3376, DOI 10.17487/RFC3376, October 2002, <https://www.rfc-editor.org/info/rfc3376>.

[RFC3740] Hardjono, T. and B. Weis, "The Multicast Group Security Architecture", RFC 3740, DOI 10.17487/RFC3740, March 2004, <https://www.rfc-editor.org/info/rfc3740>.

[RFC3810] Vida, R., Ed. and L. Costa, Ed., "Multicast Listener Discovery Version 2 (MLDv2) for IPv6", RFC 3810, DOI 10.17487/RFC3810, June 2004, <https://www.rfc-editor.org/info/rfc3810>.

[RFC4046] Baugher, M., Canetti, R., Dondeti, L., and F. Lindholm, "Multicast Security (MSEC) Group Key Management Architecture", RFC 4046, DOI 10.17487/RFC4046, April 2005, <https://www.rfc-editor.org/info/rfc4046>.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, DOI 10.17487/RFC4301, December 2005, <https://www.rfc-editor.org/info/rfc4301>.

[RFC4535] Harney, H., Meth, U., Colegrove, A., and G. Gross, "GSAKMP: Group Secure Association Key Management Protocol", RFC 4535, DOI 10.17487/RFC4535, June 2006, <https://www.rfc-editor.org/info/rfc4535>.

[RFC4944] Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks", RFC 4944, DOI 10.17487/RFC4944, September 2007, <https://www.rfc-editor.org/info/rfc4944>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", FYI 36, RFC 4949, DOI 10.17487/RFC4949, August 2007, <https://www.rfc-editor.org/info/rfc4949>.

[RFC6282] Hui, J., Ed. and P. Thubert, "Compression Format for IPv6 Datagrams over IEEE 802.15.4-Based Networks", RFC 6282, DOI 10.17487/RFC6282, September 2011, <https://www.rfc-editor.org/info/rfc6282>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <https://www.rfc-editor.org/info/rfc6347>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://www.rfc-editor.org/info/rfc6749>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <https://www.rfc-editor.org/info/rfc7228>.

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <https://www.rfc-editor.org/info/rfc7390>.

Appendix A. 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.

- 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 ligthing 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.

- 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.

- 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 latters 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.

- 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".

- 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.

- 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 enpoint 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.

- Security Common Context: the OSCORE Security Common Context associated to the joined multicast group (see Section 3).

- 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".

- 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:

- The joining endpoint acts as Client;
- The Group Manager acts as Resource Server, with different CoAP resources for different multicast groups it is responsible for;
- 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 synchonize 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

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

Table of Contents

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:

- Replacement of TLS by DTLS and HTTP by CoAP, resulting in:
  - DTLS-secured CoAP sessions between EST-coaps client and EST-coaps server.
- 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.
- 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

- DTLS with a previously issued client certificate (e.g., an existing certificate issued by the EST CA);
- 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.

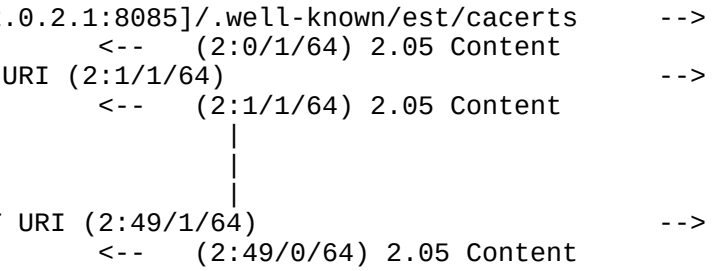
The following is an example of a valid /cacerts exchange.

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.



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: July 22, 2017

P. van der Stok
consultant
A. Bierman
YumaWorks
M. Veillette
Trilliant Networks Inc.
A. Pelov
Acklio

January 18, 2017

CoAP Management Interface
draft-vanderstok-core-comi-11

Abstract

This document describes a network management interface for constrained devices and networks, called CoAP Management Interface (CoMI). The Constrained Application Protocol (CoAP) is used to access data resources specified in YANG, or SMIv2 converted to YANG. CoMI uses the YANG to CBOR mapping and converts YANG identifier strings to numeric identifiers for payload size reduction. CoMI extends the set of YANG based protocols, NETCONF and RESTCONF, with the capability to manage constrained devices and networks.

Note

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

Status of This Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at 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 July 22, 2017.

Table of Contents

1. Introduction

The Constrained Application Protocol (CoAP) [RFC7252] is designed for Machine to Machine (M2M) applications such as smart energy 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. The 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 the draft [I-D.ietf-netconf-restconf] which describes a REST-like interface called RESTCONF, which uses HTTP methods to access structured data defined in YANG.

The use of standardized data sets, specified in a standardized language such as YANG, promotes interoperability between devices and applications from different manufacturers. A large amount of Management Information Base (MIB) [mibreg] specifications already exists for monitoring purposes. This data can be accessed in RESTCONF or CoMI if the server converts the SMIv2 modules to YANG, using the mapping rules defined in [RFC6643].

CoMI and RESTCONF are intended to work in a stateless client-server fashion. They use a single round-trip to complete a single editing transaction, where NETCONF needs up to 10 round trips.

To promote small packets, CoMI uses a YANG to CBOR mapping [I-D.ietf-core-yang-cbor] and numeric identifiers [I-D.ietf-core-sid] to minimize CBOR payloads and URI length.

1.1. Terminology

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

Readers of this specification should be familiar with all the terms and concepts discussed in [RFC3410], [RFC3416], and [RFC2578].

The following terms are defined in the NETCONF protocol [RFC6241]: client, configuration data, datastore, and server.

The following terms are defined in the YANG data modelling language [RFC7950]: anydata, anyxml, container, data node, key, key leaf, leaf, leaf-list, and list.

The following terms are defined in RESTCONF protocol [I-D.ietf-netconf-restconf]: data resource, datastore resource, edit operation, query parameter, and target resource.

The following terms are defined in this document:

data node instance: An instance of a data node specified in a YANG module present in the server. The instance is stored in the memory of the server.

Notification instance: An instance of a schema node of type notification, specified in a YANG module present in the server. The instance is generated in the server at the occurrence of the corresponding event and appended to a stream.

YANG schema item identifier: Numeric identifier which replaces the name identifying a YANG item (see section 6.2 of [RFC7950]) (anydata, anyxml, data node, RPC, Action, Notification, Identity, Module name, Submodule name, Feature).

list instance identifier: Handle used to identify a YANG data node that is an instance of a YANG "list" specified with the values of the key leaves of the list.

single instance identifier: Handle used to identify a specific data node which can be instantiated only once. This includes data nodes defined at the root of a YANG module or submodule 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.

data node value: Value assigned to a data node instance. Data node values are encoded based on the rules defined in section 4 of [I-D.ietf-core-yang-cbor].

set of data node instances: Represents the payload of CoAP methods when a collection is sent or returned. There are two possibilities, dependent on Request context:

1. CBOR array of pair(s) <instance identifier, data node value >

2. CBOR map of pair(s) <instance identifier, data node value >

TODO: Reduce to one, if possible

The following list contains the abbreviations used in this document.

SID: YANG Schema Item iDentifier.

2. CoMI Architecture

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

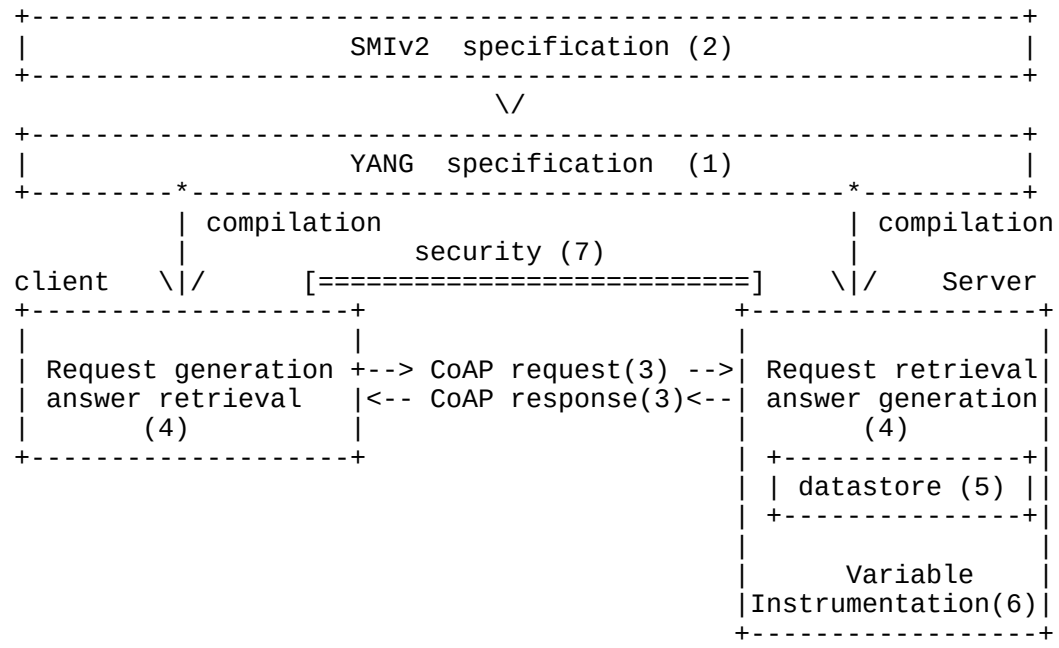


Figure 1: Abstract CoMI architecture

Figure 1 is a high level representation of the main elements of the CoAP management architecture. A client sends requests as payload in packets over the network to a managed constrained node.

The different numbered components of Figure 1 are discussed according to component number.

(1) YANG specification: contains a set of named and versioned modules.

(2) SMIv2 specification: A named module specifies a set of variables and "conceptual tables". There is an algorithm to translate SMIv2 specifications to YANG specifications.

(3) CoMI messages: The CoMI client sends request messages to and receives response messages from the CoMI server.

(4) Retrieval, generation: The server and client parse the CoMI request/response and identify the corresponding instances in the datastore based on YANG specification.

(5) Datastore: The store is composed of two parts: Operational state and Configuration datastore. Datastore also supports RPCs and event streams.

(6) Variable instrumentation: This code depends on implementation of drivers and other node specific aspects.

(7) Security: The server MUST prevent unauthorized users from reading or writing any data resources. CoMI relies on security protocols such as DTLS [RFC6347] to secure CoAP communication.

2.1. Major differences between RESTCONF and CoMI

CoMI uses CoAP/UDP as transport protocol and CBOR as payload format [I-D.ietf-core-yang-cbor]. RESTCONF uses HTTP/TCP as transport protocol and JSON [RFC7159] or XML [XML] as payload formats. CoMI encodes YANG identifier strings as numbers, where RESTCONF does not.

CoMI uses the methods FETCH and iPATCH, not used by RESTCONF. RESTCONF uses the HTTP methods HEAD, and OPTIONS, which are not used by CoMI.

CoMI servers cannot change the order of user-ordered data. CoMI does not support insert-mode (first, last, before, after) and insertion-point (before, after) which are supported by RESTCONF. Many CoAP servers will not support date and time functions. For that reason CoMI does not support the start, stop options for events.

CoMI servers only implement the efficient "trim" mode for default values.

The CoMI servers do not support the following RESTCONF functionality:

- The "fields" query parameter to query multiple instances.
- The ’filter’ query that involves XML parsing, ’content’, and ’depth’, query parameters.

2.2. Compression of YANG identifiers

In the YANG specification items are identified with a name string. In order to significantly reduce the size of identifiers used in CoMI, numeric object identifiers are used instead of these strings. The specific encoding of the object identifiers is not hard-wired in the protocol.

Examples of object identifier encoding formats are described in [I-D.ietf-core-sid].

3. Example syntax

This section presents the notation used for the examples. The YANG specifications that are used throughout this document are shown in Appendix A. The example specifications are taken over from existing modules and annotated with SIDs. The values of the SIDs are taken over from [yang-cbor].

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

A YANG (item identifier, item value) pair is mapped to a CBOR (key, value) pair. The YANG item value is encoded as specified in [I-D.ietf-core-yang-cbor]. The YANG item identifier can be a SID (single node identifier) or a CBOR array with the structure [SID, key1, key2] (list node identifier), where SID is a list identifier and the key values specify the list instance. The YANG item value can be any CBOR major type.

Delta encoding is used for the SIDs. The notation +n is used when the SID has the value PREC+n where PREC is the SID of the parent container, or PREC is the SID of the preceding entity in a CBOR array.

In all examples the resource path in the URI is expressed as a SID, represented as a base64 number. SIDs in the payload are represented as decimal numbers.

4. CoAP Interface

In CoAP a group of links can constitute a Function Set.

TODO: what will happen to term Function Set ?

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

The path prefix /c has resources accessible with the following three paths:

/c: YANG-based data with path "/c" and using CBOR content encoding format. This path represents a datastore resource which contains

YANG data resources as its descendant nodes. The data nodes are identified with their SID with format /c/SID.

/c/mod.uri: URI identifying the location of the server module information, with path "/c/mod.uri" and CBOR content format. This YANG data is encoded with plain identifier strings, not YANG encoded values. An Entity Tag MUST be maintained for this resource by the server, which MUST be changed to a new value when the set of YANG modules in use by the server changes.

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

The mapping of YANG data node instances to CoMI resources is as follows: A YANG module describes a set of data trees composed of YANG data nodes. Every data node of the YANG modules loaded in the CoMI server represents a resource of the datastore container (e.g. /c/<sid>

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

TODO; reference to fetch and patch content formats.

The profile of the management function set, with IF=core.c, is shown in the table below, following the guidelines of [I-D.ietf-core-interfaces]:

name	path	rt	Data Type
Management	/c	core.c	n/a
Data	/c	core.c.data	application/cbor
Module Set URI	/c/mod.uri	core.c.moduri	application/cbor
Events	/c/s	core.c.stream	application/cbor

5. /c Function Set

The /c Function Set provides a CoAP interface to manage YANG servers.

The methods used by CoMI are:

Operation	Description
GET	Retrieve the datastore resource or a data resource
FETCH	Retrieve (partial) data resource(s)
POST	Create a data resource, invoke RPC
PUT	Create or replace a data resource
iPATCH	Idem-potently create, replace, and delete data resource(s) (partially)
DELETE	Delete a data resource

There is one query parameters for the GET, PUT, POST, and DELETE methods.

Query Parameter	Description
k	Select an instance of a list node

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

5.1. Using the 'k' query parameter

The "k" (key) parameter specifies the instance of a list node. The SID in the URI is followed by the (?k=key1, key2,..). Where SID identifies a list node, and key1, key2 are the values of the key leaves that specify an instance of the list.

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

YANG datatype	Binary representation	Text representation
uint8,uint16,unit32, uint64	CBOR unsigned integer	int_to_text(number)
int8, int16,int32, int64	CBOR negative integer	base64 (CBOR representation)
decimal64	CBOR decimal fractions	base64 (CBOR representation
string	CBOR text or string	text
boolean	CBOR false or true	"0" or "1"
enumeration	CBOR unsigned integer	int_to_text (number)
bits	CBOR byte string	base64 (CBOR representation)
binary	CBOR byte string	base64 (binary value)
identityref	CBOR unsigned integer	int_to_text (number)
union		base64 (CBOR representation)
List instance identifier	CBOR unsigned integer	base64 (CBOR representation)
List instance identifier	CBOR array	Base64 (CBOR representation)

5.2. Data Retrieval

One or more data node instances 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 [I-D.ietf-core-etch].

It is possible that the size of the payload is too large to fit in a single message. In the case that management data is bigger than the maximum supported payload size, the Block mechanism from [RFC7959] is used, as explained in more detail in Section 7.

CoMI uses the FETCH payload for retrieving a subset of the datastore.

There are two additional query parameters for the GET and FETCH methods.

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

5.2.1. Using the 'c' query parameter

The 'c' (content) parameter controls how descendant nodes of the requested data nodes will be processed in the reply.

The allowed values are:

Value	Description
c	Return only configuration descendant data nodes
n	Return only non-configuration descendant data nodes
a	Return all descendant data nodes

This parameter is only allowed for GET and FETCH methods on datastore and data resources. A 4.00 Bad Request error is returned if used for other methods or resource types.

If this query parameter is not present, the default value is "a".

5.2.2. Using the 'd' query parameter

The "d" (with-defaults) parameter controls how the default values of the descendant nodes of the requested data nodes will be processed.

The allowed values are:

Value	Description
a	All data nodes are reported| Defined as 'report-all' in section 3.1 of [RFC6243].
t	Data nodes set to the YANG default are not reported. Defined as 'trim' in section 3.2 of [RFC6243].

If the target of a GET or FETCH method is a data node that represents a leaf that has a default value, and the leaf has not been given a value yet, the server MUST return the leaf.

If the target of a GET method is a data node that represents a container or list that has any child resources with default values, for the child resources that have not been given value yet, the server MUST not return the child resource if this query parameter is set to 't' and MUST return the child resource if this query parameter is set to 'a'.

If this query parameter is not present, the default value is 't'.

5.2.3. GET

A request to read the values of a data node instance is sent with a confirmable CoAP GET message. A single instance identifier is specified in the URI path prefixed with /c.

FORMAT:

```
GET /c/<instance identifier>

2.05 Content (Content-Format: application/cbor)
<data node value>
```

The returned payload is composed of all the children associated with the specified data node instance.

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.2.3.1. GET Examples

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

state/clock/current-datetime is 1719, encoded in base64 this yields a3. The answer to the request returns a <value>, transported as a single CBOR string item.

```
REQ: GET example.com/c/a3

RES: 2.05 Content (Content-Format: application/cbor)
"2014-10-26T12:16:31Z"
```

For example, the GET of the clock node (ID = 1717; base64: a1), sent by the client, results in the following returned value sent by the server, transported as a CBOR map containing 2 pairs:

```
REQ: GET example.com/c/a1

RES: 2.05 Content (Content-Format: application/cbor)
{
      +2 : "2014-10-26T12:16:51Z",   / ID 1719 /
      +1 : "2014-10-21T03:00:00Z"    / ID 1718 /
}
```

A "list" node can have multiple instances. Accordingly, the returned payload of GET is composed of all the instances associated with the selected list node.

For example, look at the example in Appendix A.3. The GET of the /interfaces/interface/ (with identifier 1533, base64: X9) results in the following returned payload, transported as a CBOR array with 2 elements.

```
REQ: GET example.com/c/X9

RES: 2.05 Content (Content-Format: application/cbor)
[
 {+4 : "eth0",                / name  (ID 1537) /
  +1 : "Ethernet adaptor",    / description (ID 1534) /
  +5 : 1179,                  / type, (ID 1538) identity /
                              / ethernetCsmacd (ID 1179) /
  +2 : true                   / enabled ( ID 1535) /
 },
 {+4 : "eth1",                / name (ID 1537) /
  +1 : "Ethernet adaptor",    / description (ID 1534) /
  +5 : 1179,                  / type, (ID 1538) identity /
                              / ethernetCsmacd (ID 1179) /
  +2 : false                  / enabled /
  }
]
```

It is equally possible to select a leaf of one instance of a list or a complete instance container with GET. The instance identifier is the numeric identifier of the list followed by the specification of the values for the key leaves that uniquely identify the list instance. The instance identifier looks like: SID?k=key-value. The key of "interface" is the "name" leaf. The example below requests the description leaf of the instance with name="eth0" (ID=1534, base64: X-). The value of the description leaf is returned.

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

RES: 2.05 Content (Content-Format: application/cbor)
"Ethernet adaptor"
```

5.2.4. FETCH

The FETCH is used to retrieve a list of data node values. The FETCH Request payload contains a CBOR list of instance identifiers.

```
FORMAT:
    FETCH /c/ Content-Format (application/YANG-fetch+cbor)
    <CBOR array of instance identifiers>

    2.05 Content (Content-Format: application/YANG-patch+cbor)
    <CBOR array of data node values>
```

The instance identifier is a SID or a CBOR array containing the SID followed by key values that identify the list instance (sec 5.13.1 of [I-D.ietf-core-yang-cbor]. In the payload of the returned data node

values, delta encoding is used as described in [I-D.ietf-core-yang-cbor].

5.2.4.1. FETCH examples

The example uses the current-datetime leaf and the interface list from Appendix A.1. In the following example the value of current-datetime (ID 1719)and the interface list (ID 1533) instance identified with name="eth0" are queried.

```
REQ:  FETCH /c Content-Format (application/YANG-fetch+cbor)
      [ 1719,            / ID 1719 /
      [-186, "eth0"]   / ID 1533 with name = "eth0" /
      ]

RES:  2.05 Content Content-Format (application/YANG-patch+cbor)
[
  "2014-10-26T12:16:31Z",
 {
   +4 : "eth0",                / name (ID 1537) /
   +1 : "Ethernet adaptor",    / description (ID 1534) /
   +5 : 1179,                  / type (ID 1538), identity /
                               / ethernetCsmacd (ID 1179) /
   +2 : true                   / enabled (ID 1535) /
  }
]
```

TODO: align with future FETCH content format.

5.3. Data Editing

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

5.3.1. Data Ordering

A CoMI server SHOULD preserve the relative order of all user-ordered list and leaf-list entries that are received in a single edit request. These YANG data node types are encoded as arrays so messages will preserve their order.

5.3.2. POST

Data resources are created with the POST method. The CoAP POST operation is used in CoMI for creation of data resources and the invocation of "ACTION" and "RPC" resources. Refer to Section 5.6 for details on "ACTION" and "RPC" resources.

A request to create the values of an instance of a container or leaf is sent with a confirmable CoAP POST message. A single SID is specified in the URI path prefixed with /c.

FORMAT:

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

2.01 Created (Content-Format: application/cbor)
```

If the data resource already exists, then the POST request MUST fail and a "4.09 Conflict" status-line MUST be returned

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.3.2.1. Post example

The example uses the interface list from Appendix A.1. Example is creating a new version of the container interface (ID = 1533):

```
REQ: POST /c/X9 Content-Format(application/cbor)
      {
        +4 : "eth0",              / name (ID 1537) /
        +1 : "Ethernet adaptor", / description (ID 1534) /
        +5 : 1179,               / type (ID 1538), identity /
                                 / ethernetCsmacd (ID 1179) /
        +2 : true                / enabled (ID 1535) /
      }

RES: 2.01 Created (Content-Format: application/cbor)
```

5.3.3. PUT

Data resource instances are created or replaced with the PUT method. The PUT operation is supported in CoMI. A request to set the value of a data node instance is sent with a confirmable CoAP PUT message.

FORMAT:

```
PUT /c/<instance identifier> Content-Format(application/cbor)
<data node value>

2.01 Created
```

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.3.3.1. PUT example

The example uses the interface list from Appendix A.1. Example is renewing an instance of the list interface (ID = 1533) with key name="eth0":

```
REQ:  PUT /c/X9?k="eth0" Content-Format(application/cbor)
      {
        +4 : "eth0",              / name (ID 1537) /
        +1 : "Ethernet adaptor", / description (ID 1534) /
        +5 : 1179,                / type (ID 1538), identity /
                                  / ethernetCsmacd ( ID 1179) /
        +2 : true                 / enabled (ID 1535) /
      }
RES:  2.04 Changed
```

5.3.4. iPATCH

One or multiple data resource instances are replaced with the idempotent iPATCH method [I-D.ietf-core-etch]. A request is sent with a confirmable CoAP iPATCH message.

There are no query parameters for the iPATCH method.

The processing of the iPATCH command is specified by the CBOR payload. The CBOR patch payload describes the changes to be made to target YANG data nodes [I-D.bormann-appsawg-cbor-merge-patch]. If the CBOR patch payload contains data node instances that are not present in the target, these instances are added or silently ignored dependent of the payload information. If the target contains the specified instance, the contents of the instances are replaced with the values of the payload. Null values indicate the removal of existing values.

FORMAT:

```
iPATCH /c Content-Format(application/YANG-patch+cbor)
<set of data node instances>

2.04 Changed
```

5.3.4.1. iPATCH example

The example uses the interface list from Appendix A.3, and the timezone-utc-offset leaf from Appendix A.1. In the example one leaf (timezone-utc-offset) and one container (interface) instance are changed.

```
REQ: iPATCH /c Content-Format(application/YANG-patch+cbor)
[
  [1533, "eth0"] ,               / interface (ID = 1533) /
    {
      +4 : "eth0",               / name (ID 1537) /
      +1 : "Ethernet adaptor",   / description (ID 1534) /
      +5 : 1179,                 / type (ID 1538), identity /
                                 / ethernetCsmacd (ID 1179) /
      +2 : true                  / enabled (ID 1535) /
    },
  +203 , 60          / timezone-utc-offset (delta = 1736-1533) /
]

RES: 2.04 Changed

TODO: Align with future cbor-merge-patch content format.
```

5.3.5. DELETE

Data resource instances are deleted with the DELETE method. The RESTCONF DELETE operation is supported in CoMI.

FORMAT:

```
    Delete /c/<instance identifier>

    2.02 Deleted
```

The instance identifier is a SID or a SID followed by the "k" query parameter.

5.3.5.1. DELETE example

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

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

RES:   2.02 Deleted
```

5.4. Full Data Store access

The methods GET, PUT, POST, and DELETE can be used to return, replace, create, and delete the whole data store respectively.

```
FORMAT:
    GET /c
    2.05 Content (Content-Format: application/cbor)
    <array of data node instances>

   PUT /c
   (Content-Format: application/cbor)
    <array of data node instances>
   2.04 Changed

   POST /c
   (Content-Format: application/cbor)
    <array of data node instances>
   2.01 Created

   DELETE /c
   2.02 Deleted
```

The array of data node instances represents an array of all root nodes in the data store after the PUT, POST and GET method invocations.

5.4.1. Full Data Store examples

The example uses the interface list and the clock container from Appendix A.3. Assume that the data store contains two root objects: the list interface (ID 1533) with one instance and the container Clock (ID 1717). After invocation of GET an array with these two objects is returned:

```
RQ:  GET /c
RES: 2.05 Content Content-Format (application/YANG-patch+cbor)
[
  {1717:
     { +2: "2016-10-26T12:16:31Z", / current-datetime (ID 1719) /
       +1: "2014-10-05T09:00:00Z"  / boot-datetime (ID 1718) /
     },
  -186:                           / clock (ID 1533) /
     {
       +4 : "eth0",               / name (ID 1537) /
       +1 : "Ethernet adaptor",   / description (ID 1534) /
       +5 : 1179,                 / type (ID 1538), identity: /
                                  / ethernetCsmacd (ID 1179) /
       +2 : true                  / enabled (ID 1535) /
      }
    }
]
```

5.5. Notify functions

Notification by the server to a selection of clients when an event occurs in the server is an essential function for the management of servers. CoMI allows events specified in YANG [RFC5277] to be notified to a selection of requesting clients. The server appends newly generated events to a stream. There is one, so-called "default", stream in a CoMI server. The /c/s resource identifies the default stream. The server MAY create additional stream resources. When a CoMI server generates an internal event, it is appended to the chosen stream, and the content of a notification instance is ready to be sent to all CoMI clients which observe the chosen stream resource.

Reception of generated notification instances is enabled with the CoAP Observe [RFC7641] function. The client subscribes to the notifications by sending a GET request with an "Observe" option, specifying the /c/s resource when the default stream is selected.

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

```
FORMAT:
  Get /<stream-resource>
      Content-Format(application/YANG-patch+cbor) Observe(0)

2.05 Content Content-Format(application/YANG-patch+cbor)
<set of data node instances>
```

5.5.1. Notify Examples

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

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

RES:  2.05 Content Content-Format(application/YANG-patch+cbor)
                        Observe(12) Token(0x93)
{
   60010 :                  / example-port-fault (ID 60010) /
    {
      +1 : "0/4/21",       / port-name (ID 60011) /
      +2 : "Open pin 2"    / port-fault (ID 60012) /
    },
   60010 :                  / example-port-fault (ID 60010) /
    {
      +1 : "1/4/21",       / port-name (ID 60011) /
      +2 : "Open pin 5"    / port-fault (ID 60012) /
    }
}
```

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

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

5.6. RPC statements

The YANG "action" and "RPC" statements specify the execution of a Remote procedure Call (RPC) in the server. It is invoked using a POST method to an "Action" or "RPC" resource instance. The Request payload contains the values assigned to the input container when specified with the action station. The Response payload contains the values of the output container when specified.

The returned success response code is 2.05 Content.

```
FORMAT:
 POST /c/<instance identifier>
           Content-Format(application/YANG-patch+cbor)
<input node value>

2.05 Content Content-Format (application/YANG-patch+cbor)
<output node value>
```

There "k" query parameter is allowed for the POST method when used for an action invocation.

5.6.1. RPC Example

The example is based on the YANG action specification of Appendix A.2. A server list is specified and the action "reset" (ID 60002, base64: Opq), that is part of a "server instance" with key value "myserver", is invoked.

```
REQ:  POST /c/Opq?k="myserver"
              Content-Format(application/YANG-patch+cbor)
{
  +1 : "2016-02-08T14:10:08Z09:00" / reset-at (ID 60003) /
}

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

6. Access to MIB Data

Appendix A.5 shows a YANG module mapped from the SMI specification "IP-MIB" [RFC4293]. The following example shows the "ipNetToPhysicalEntry" list with 2 instances, using diagnostic notation without delta encoding.

```
{
   60021 :                    / list ipNetToPhysicalEntry /
   [
     {
       60022 : 1,             / ipNetToPhysicalIfIndex /
       60023 : 1,             / ipNetToPhysicalNetAddressType: ipv4 /
       60024 : h'0A000033',   / ipNetToPhysicalNetAddress /
       60025 : h'00000A01172D',/ ipNetToPhysicalPhysAddress /
       60026 : 2333943,       / ipNetToPhysicalLastUpdated /
       60027 : 4,             / ipNetToPhysicalType: static /
       60028 : 1,             / ipNetToPhysicalState: reachable /
       60029 : 1              / ipNetToPhysicalRowStatus: active /
     },
     {
       60022 : 1,             / ipNetToPhysicalIfIndex /
       60023 : 1,             / ipNetToPhysicalNetAddressType: ipv4 /
       60024 : h'09020304',   / ipNetToPhysicalNetAddress  /
       60025 : h'00000A36200A',/ ipNetToPhysicalPhysAddress /
       60026 : 2329836,       / ipNetToPhysicalLastUpdated /
       60027 : 3,             / ipNetToPhysicalType: dynamic /
       60028 : 6,             / ipNetToPhysicalState: unknown /
       60029 : 1              / ipNetToPhysicalRowStatus: active /
     }
   ]
}
```

The IPv4 addresses A.0.0.33 and 9.2.3.4 are encoded in CBOR as h'0A000033' and h'09020304' respectively. In the following example exactly one instance is requested from the ipNetToPhysicalEntry (ID 60021, base64: Oz1). The h'09020304' value is encoded in base64 as AJAgME.

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

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

RES: 2.05 Content (Content-Format: application/YANG-patch+cbor)
{
   +1 : 1,                  / ( SID 60022 ) /
   +2 : 1,                  / ( SID 60023 ) /
   +3 : h'09020304',        / ( SID 60024 ) /
   +4 : h'00000A36200A',    / ( SID 60025 ) /
   +5 : 2329836,            / ( SID 60026 ) /
   +6 : 3,                  / ( SID 60027 ) /
   +7 : 6,                  / ( SID 60028 ) /
   +8 : 1                   / ( SID 60029 ) /
}
```

7. Use of Block

The CoAP protocol provides reliability by acknowledging the UDP datagrams. However, when large pieces of text need to be transported the datagrams get fragmented, thus creating constraints on the resources in the client, server and intermediate routers. The block option [RFC7959] allows the transport of the total payload in individual blocks of which the size can be adapted to the underlying transport sizes such as: (UDP datagram size ~64KiB, IPv6 MTU of 1280, IEEE 802.15.4 payload of 60-80 bytes). Each block is individually acknowledged to guarantee reliability.

Notice that the Block mechanism splits the data at fixed positions, such that individual data fields may become fragmented. Therefore, assembly of multiple blocks may be required to process the complete data field.

Beware of race conditions. Blocks are filled one at a time and care should be taken that the whole data representation is sent in multiple blocks sequentially without interruption. In the server, values are changed, lists are re-ordered, extended or reduced. When these actions happen during the serialization of the contents of the variables, 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 actual number of items. It may be advisable to use CBOR maps or CBOR arrays of undefined length which are foreseen for data streaming purposes.

8. Resource Discovery

The presence and location of (path to) the management data are discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c" [RFC6690].

Upon success, the return payload will contain the root resource of the management data. It is up to the implementation to choose its root resource, but it is recommended that the value "/c" is used, where possible. The example below shows the discovery of the presence and location of management data.

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

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

Management objects MAY be discovered with the standard CoAP resource discovery. The implementation can add the encoded values of the object identifiers to /.well-known/core with rt="core.c.data". The available objects identified by the encoded values can be discovered by sending a GET request to "/.well-known/core" including a resource type (RT) parameter with the value "core.c.data". Upon success, the return payload will contain the registered encoded values and their location. The example below shows the discovery of the presence and location of management data.

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

RES: 2.05 Content </c/BaAiN>; rt="core.c.data",
</c/CF_fA>; rt="core.c.data"
```

Lists of encoded values may become prohibitively long. It is discouraged to provide long lists of objects on discovery. Therefore, it is recommended that details about management objects are discovered by reading the YANG module information stored in for example the "ietf-comi-yang-library" module [I-D.veillette-core-cool-library]. The resource "/c/mod.uri" is used to retrieve the location of the YANG module library.

The module list can be stored locally on each server, or remotely on a different server. The latter is advised when the deployment of many servers are identical.

```
Local in example.com server:

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

RES: 2.05 Content (Content-Format: application/cbor)
{
  "mod.uri" : "example.com/c/modules"
}


Remote in example-remote-server:

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

RES: 2.05 Content (Content-Format: application/cbor)
{
  "moduri" : "example-remote-server.com/c/group17/modules"
}
```

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

9. Error Handling

In case a request is received which cannot be processed properly, the CoMI server MUST return an error message. This error message MUST contain a CoAP 4.xx or 5.xx response code, and SHOULD include additional information in the payload.

Such an error message payload is a text string, using the following structure:

```
CoMI error: xxxx "error text"
```

The characters xxxx represent one of the values from the table below, and the OPTIONAL "error text" field contains a human readable explanation of the error.

CoMI Error Code	CoAP Error Code	Description
0	4.xx	General error
1	4.13	Request too big
2	4.00	Response too big
3	4.00	Unknown identifier
4	4.00	Invalid value
5	4.05	Attempt to write read-only variable
6	5.01	No access
7	4.00	Wrong type
8	4.15	Unknown encoding
9	4.0	Wrong value
10	4.0	Not created
11	4.04	Resource unavailable
12	4.01	Authorization error
13	4.0	Bad attribute
14	4.0	Unknown attribute
15	4.0	Missing attribute

The CoMI error codes are motivated by the error-status values defined in [RFC3416], and the error tags defined in [I-D.ietf-netconf-restconf].

10. Security Considerations

For secure network management, it is important to restrict access to configuration variables only to authorized parties. This requires integrity protection of both requests and responses, and depending on the application encryption.

CoMI re-uses the security mechanisms already available to CoAP as much as possible. This includes DTLS [RFC6347] for protected access to resources, as well suitable authentication and authorization mechanisms.

Among the security decisions that need to be made are selecting security modes and encryption mechanisms (see [RFC7252]). This requires a trade-off, as the NoKey mode gives no protection at all, but is easy to implement, whereas the X.509 mode is quite secure, but may be too complex for constrained devices.

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

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

11. IANA Considerations

'rt="core.c"' needs registration with IANA.

'rt="core.c.data"' needs registration with IANA.

'rt="core.c.moduri"' needs registration with IANA.

'rt="core.c.stream"' needs registration with IANA.

Content types to be registered:

- application/YANG-patch+cbor
- application/YANG-fetch+cbor

12. Acknowledgements

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

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

Timothy Carey has provided the text for Appendix B.

The draft has benefited from comments (alphabetical order) by Rodney Cummings, Dee Denteneer, Esko Dijk, Michael van Hartskamp, Juergen

Schoenwaelder, Anuj Sehgal, Zach Shelby, Hannes Tschofenig, Michael Verschoor, and Thomas Watteyne.

13. Changelog

Changes from version 00 to version 01

- Focus on MIB only
- Introduced CBOR, JSON, removed BER
- defined mappings from SMI to xx
- Introduced the concept of addressable table rows

Changes from version 01 to version 02

- Focus on CBOR, used JSON for examples, removed XML and EXI
- added uri-query attributes mod and con to specify modules and contexts
- Definition of CBOR string conversion tables for data reduction
- use of Block for multiple fragments
- Error returns generalized
- SMI - YANG - CBOR conversion

Changes from version 02 to version 03

- Added security considerations

Changes from version 03 to version 04

- Added design considerations section
- Extended comparison of management protocols in introduction
- Added automatic generation of CBOR tables
- Moved lowpan table to Appendix

Changes from version 04 to version 05

- Merged SNMP access with RESTCONF access to management objects in small devices

- Added CoMI architecture section
- Added RESTCONf NETMOD description
- Rewrote section 5 with YANG examples
- Added server and payload size appendix
- Removed Appendix C for now. It will be replaced with a YANG example.

Changes from version 04 to version 05

- Extended examples with hash representation
- Added keys query parameter text
- Added select query parameter text
- Better separation between specification and instance
- Section on discovery updated
- Text on rehashing introduced
- Elaborated SMI MIB example
- YANG library use described
- use of BigEndian/LittleEndian in Hash generation specified

Changes from version 05 to version 06

- Hash values in payload as hexadecimal and in URL in base64 numbers
- Streamlined CoMI architecture text
- Added select query parameter text
- Data editing optional
- Text on Notify added
- Text on rehashing improved with example

Changes from version 06 to version 07

- reduced payload size by removing JSON hierarchy

- changed rehash handling to support small clients
- added LWM2M comparison
- Notification handling as specified in YANG
- Added Patch function
- Rehashing completely reviewed
- Discover type of YANG name encoding
- Added new resource types
- Read-only servers introduced
- Multiple updates explained

Changes from version 07 to version 08

- Changed YANG Hash algorithm to use module name instead of prefix
- Added rehash bit to allow return values to identify rehashed nodes in the response
- Removed /c/mod.set resource since this is not needed
- Clarified that YANG Hash is done even for unimplemented objects
- YANG lists transported as CBOR maps of maps
- Adapted examples with more CBOR explanation
- Added CBOR code examples in new appendix
- Possibility to use other than default stream
- Added text and examples for Patch payload
- Repaired some examples
- Added appendices on hash clash probability and hash clash storage overhead

Changes from version 08 to version 09

- Removed hash and YANG to CBOR sections

- removed hashes from examples.
- Added RPC
- Added content query parameter.
- Added default handling.
- Listed differences with RESTCONF

Changes from version 09 to version 10. This is the merge of cool-01 with comi-09.

- Merged with CoOL SIDs
- Introduced iPATCH, PATCH and FETCH
- Update of LWM2M comparison
- Added appendix with module examples
- Removed introductory text
- Removed references

Changes from version 10 to version 11

- Introduction streamlined
- Error codes streamlined
- Examples updated to latest SID values
- Update of the YANG specifications in the appendix

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, <http://www.rfc-editor.org/info/rfc2119>.

[RFC5277] Chisholm, S. and H. Trevino, "NETCONF Event Notifications", RFC 5277, DOI 10.17487/RFC5277, July 2008, <http://www.rfc-editor.org/info/rfc5277>.

[RFC6243] Bierman, A. and B. Lengyel, "With-defaults Capability for NETCONF", RFC 6243, DOI 10.17487/RFC6243, June 2011, <http://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, <http://www.rfc-editor.org/info/rfc7049>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[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>.

[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>.

[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>.

[I-D.ietf-netconf-restconf] Bierman, A., Bjorklund, M., and K. Watsen, "RESTCONF Protocol", draft-ietf-netconf-restconf-18 (work in progress), October 2016.

[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.bormann-appsawg-cbor-merge-patch] Bormann, C. and P. Stok, "CBOR Merge Patch", draft-bormann-appsawg-cbor-merge-patch-00 (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-03 (work in progress), October 2016.

14.2. Informative References

[RFC2578] McCloghrie, K., Ed., Perkins, D., Ed., and J. Schoenwaelder, Ed., "Structure of Management Information Version 2 (SMIv2)", STD 58, RFC 2578, DOI 10.17487/RFC2578, April 1999, <http://www.rfc-editor.org/info/rfc2578>.

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, DOI 10.17487/RFC3410, December 2002, <http://www.rfc-editor.org/info/rfc3410>.

[RFC3416] Presuhn, R., Ed., "Version 2 of the Protocol Operations for the Simple Network Management Protocol (SNMP)", STD 62, RFC 3416, DOI 10.17487/RFC3416, December 2002, <http://www.rfc-editor.org/info/rfc3416>.

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

[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>.

[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>.

[RFC6643] Schoenwaelder, J., "Translation of Structure of Management Information Version 2 (SMIv2) MIB Modules to YANG Modules", RFC 6643, DOI 10.17487/RFC6643, July 2012, <http://www.rfc-editor.org/info/rfc6643>.

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

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

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

[RFC7317] Bierman, A. and M. Bjorklund, "A YANG Data Model for System Management", RFC 7317, DOI 10.17487/RFC7317, August 2014, <http://www.rfc-editor.org/info/rfc7317>.

[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-07 (work in progress), December 2016.

[I-D.ietf-core-sid] Somaraju, A., Veillette, M., Pelov, A., Turner, R., and A. Minaburo, "YANG Schema Item iDentifier (SID)", draft-ietf-core-sid-00 (work in progress), October 2016.

[I-D.veillette-core-cool] Veillette, M., Pelov, A., Somaraju, A., Turner, R., and A. Minaburo, "Constrained Objects Language", draft-veillette-core-cool-02 (work in progress), July 2016.

[I-D.veillette-core-cool-library] Veillette, M., "Constrained YANG Module Library", draft-veillette-core-cool-library-00 (work in progress), August 2016.

[XML] "Extensible Markup Language (XML)", Web http://www.w3.org/xml.

[OMA] "OMA-TS-LightweightM2M-V1_0-20131210-C", Web http://technical.openmobilealliance.org/Technical/current_releases.aspx.

[OMNA] "Open Mobile Naming Authority (OMNA)", Web http://http://technical.openmobilealliance.org/Technical/technical-information/omna.

[netconfcentral] "NETCONF Central: library of YANG modules", Web http://www.netconfcentral.org/modulelist.

[mibreg] "Structure of Management Information (SMI) Numbers (MIB Module Registrations)", Web http://www.iana.org/assignments/smi-numbers/smi-numbers.xhtml/.

[yang-cbor] "yang-cbor Registry", Web https://github.com/core-wg/yang-cbor/tree/master/registry/.

Appendix A. YANG example specifications

This appendix shows 5 YANG example specifications taken over from as many existing YANG modules. The YANG modules are available from [netconfcentral]. Each YANG item identifier is accompanied by its SID shown after the "//" comment sign, taken from [yang-cbor].

A.1. ietf-system

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

```
module ietf-system {
  container system {                    // SID 1715
    container clock {                   // SID 1734
      choice timezone {
        case timezone-name {
          leaf timezone-name {          // SID 1735
            type timezone-name;
          }
        }
        case timezone-utc-offset {
          leaf timezone-utc-offset {    // SID 1736
            type int16 {
            }
          }
        }
      }
    }
    container ntp {                     // SID 1750
      leaf enabled {                    // SID 1751
        type boolean;
        default true;
      }
      list server {                     // SID 1752
        key name;
        leaf name {                     // SID 1755
          type string;
        }
        choice transport {
          case udp {
            container udp {             // SID 1757
              leaf address {            // SID 1758
                type inet:host;
              }
              leaf port {               // SID 1759
                type inet:port-number;
              }
            }
```

```
            }
          }
          leaf association-type {        // SID 1753
            type enumeration {
              enum server {
              }
              enum peer {
              }
              enum pool {
              }
            }
          }
          leaf iburst {                  // SID 1754
            type boolean;
          }
          leaf prefer {                  // SID 1756
            type boolean;
            default false;
          }
        }
      }
    container system-state {             // SID 1716
      container clock {                  // SID 1717
        leaf current-datetime {          // SID 1719
          type yang:date-and-time;
        }
        leaf boot-datetime {             // SID 1718
          type yang:date-and-time;
        }
      }
    }
  }
```

A.2. server list

Taken over from [RFC7950] section 7.15.3.

```
module example-server-farm {
  yang-version 1.1;
  namespace "urn:example:server-farm";
  prefix "sfarm";

  import ietf-yang-types {
    prefix "yang";
  }

  list server {                          // SID 60000
    key name;
    leaf name {                          // SID 60001
      type string;
    }
    action reset {                       // SID 60002
      input {
        leaf reset-at {                  // SID 60003
          type yang:date-and-time;
          mandatory true;
        }
      }
      output {
        leaf reset-finished-at {         // SID 60004
          type yang:date-and-time;
          mandatory true;
        }
      }
    }
  }
}
```

A.3. interfaces

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

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

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

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

A.4. Example-port

Notification example defined within this document.

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

A.5. IP-MIB

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

```
module IP-MIB {
  import IF-MIB {
    prefix if-mib;
  }
  import INET-ADDRESS-MIB {
    prefix inet-address;
  }
  import SNMPv2-TC {
    prefix smiv2;
  }
  import ietf-inet-types {
    prefix inet;
  }
  import yang-smi {
    prefix smi;
  }
  import ietf-yang-types {
    prefix yang;
  }

  container ip {                              // SID 60020
    list ipNetToPhysicalEntry {               // SID 60021
      key "ipNetToPhysicalIfIndex ipNetToPhysicalNetAddressType ipNetToPhysicalN
etAddress";
      leaf ipNetToPhysicalIfIndex {           // SID 60022
        type if-mib:InterfaceIndex;
```

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

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

Appendix B. Comparison with LWM2M

B.1. Introduction

CoMI and LWM2M [OMA], both, provide RESTful device management services over CoAP. Differences between the designs are highlighted in this section.

The intent of the LWM2M protocol is to provide a single protocol to control and manage IoT devices. This means the IoT device implements and uses the same LWM2M agent function for the actuation and sensing features of the IoT device as well as for the management of the IoT device. The intent of CoMI Interface as described in the Abstract section of this document is to provide management of constrained devices and devices in constrained networks using RESTCONF and YANG. This implies that the device, although reusing the CoAP protocol, would need a separate CoAP based agent in the future to control the actuation and sensing features of the device and another CoMI agent that performs the management functions.

It should be noted that the mapping of a LWM2M server to YANG is specified in [YANGlwm2m]. The converted server can be invoked with CoMI as specified in this document.

For the purposes of managing IoT devices the following points related to the protocols compare how management resources are defined, identified, encoded and updated.

B.2. Defining Management Resources

Management resources in LWM2M (LWM2M objects) are defined using a standardized number. When a new management resource is defined, either by a standards organization or a private enterprise, the management resource is registered with the Open Mobile Naming Authority [OMNA] in order to ensure different resource definitions do not use the same identifier. CoMI, by virtue of using YANG as its data modeling language, allows enterprises and standards organizations to define new management resources (YANG nodes) within YANG modules without having to register each individual management resource. Instead YANG modules are scoped within a registered name space. As such, the CoMI approach provides additional flexibility in defining management resources. Likewise, since CoMI utilizes YANG, existing YANG modules can be reused. The flexibility and reuse capabilities afforded to CoMI can be useful in management of devices like routers and switches in constrained networks. However for management of IoT devices, the usefulness of this flexibility and applicability of reuse of existing YANG modules may not be warranted. The reason is that IoT devices typically do not require complex sets of configuration or monitoring operations required by devices like a router or a switch. To date, OMA has defined approximately 15 management resources for constrained and non-constrained mobile or fixed IoT devices while other 3rd Party SDOs have defined another 10 management resources for their use in non-constrained IoT devices. Likewise, the Constrained Object Language [I-D.veillette-core-cool] which is used by CoMI when managing constrained IoT devices uses YANG schema item identifiers, which are registered with IANA, in order to define management resources that are encoded using CBOR when targeting constrained IoT Devices.

B.3. Identifying Management Resources

As LWM2M and CoMI can similarly be used to manage IoT devices, comparison of the CoAP URIs used to identify resources is relevant as the size of the resource URI becomes applicable for IoT devices in constrained networks. LWM2M uses a flat identifier structure to identify management resources and are identified using the LWM2M object’s identifier, instance identifier and optionally resource identifier (for access to and object’s attributes). For example, identifier of a device object (object id = 3) would be "/3/0" and identification of the device object’s manufacturer attribute would be "/3/0/0". Effectively LWM2M identifiers for management resources are between 4 and 10 bytes in length.

CoMI is expected to be used to manage constrained IoT devices. CoMI utilizes the YANG schema item identifier[SID] that identify the resources. CoMI recommends that IoT device expose resources to

identify the data stores and event streams of the CoMI agent. Individual resources (e.g., device object) are not directly identified but are encoded within the payload. As such the identifier of the CoMI resource is smaller (4 to 7 bytes) but the overall payload size isn't smaller as resource identifiers are encoded on the payload.

B.4. Encoding of Management Resources

LWM2M provides a separation of the definition of the management resources from how the payloads are encoded. As of the writing of this document LWM2M encodes LWM2M encodes payload data in Type-length-value (TLV), JSON or plain text formats. JSON encoding is the most common encoding scheme with TLV encoding used on the simplest IoT devices. CoMI's use of CBOR provides a more efficient transfer mechanism [RFC7049] than the current LWM2M encoding formats.

In situations where resources need to be modified, CoMI uses the CoAP PATCH operation resources only require a partial update. LWM2M does not currently use the CoAP PATCH operation but instead uses the CoAP PUT and POST operations which are less efficient.

Authors' Addresses

Peter van der Stok
consultant

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

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

Email: andy@yumaworks.com

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

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

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

Email: a@ackl.io

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

Table of Contents

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:

- client
- configuration data
- server
- state data

The following terms are defined in [RFC7950] and are not redefined here:

- data model
- 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:

- 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.

- 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.

- 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.

- 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.

- To specify the range of a data resource the YANG range statement, specified in section 9.2.4 of [RFC7950], is used.

- 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].

- The YANG units statement, specified in section 7.3.3 of [RFC7950], is used to express the units.

- The attributes of the YANG leaf need to be presented in the order: "type", possibly qualified with "range", "units", "config", "mandatory", and finally "Description".

- 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.

- In Appendix A, Yang Identifiers are modelled as strings that start with string "ID" followed by the identifier number (see module humidityID).

- 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).

- 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)/resource >= 1. The value of the identifier = (object*10**n)+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