

CoRE	Z. Shelby	
Internet-Draft	Sensinode	
Intended status: Informational	B. Frank	
Expires: November 11, 2010	SkyFoundry	
	D. Sturek	
	Pacific Gas & Electric	
	May 10, 2010	

[TOC](#)

## **Constrained Application Protocol (CoAP)**

### **draft-shelby-core-coap-01**

#### **Abstract**

This document specifies the Constrained Application Protocol (CoAP), a specialized transfer protocol for use with constrained networks and nodes for machine-to-machine applications such as smart energy and building automation. These constrained nodes often have 8-bit microcontrollers with small amounts of ROM and RAM, while networks such as 6LoWPAN often have high packet error rates and typical throughput of 10s of kbps. CoAP provides request/reply and subscribe/notify interaction models between application end-points, supports built-in resource discovery, and includes key web concepts such as URIs and RESTful methods. CoAP easily translates to HTTP for integration with the web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments.

#### **Status of this Memo**

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

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

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

This Internet-Draft will expire on November 11, 2010.

## Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

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

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

---

## Table of Contents

- [1.](#) Introduction
- [2.](#) Constrained Application Protocol
  - [2.1.](#) Interaction Model
    - [2.1.1.](#) Request messages
    - [2.1.2.](#) Notify messages
    - [2.1.3.](#) Response message
    - [2.1.4.](#) Option fields
    - [2.1.5.](#) Transaction IDs
  - [2.2.](#) Methods
    - [2.2.1.](#) GET
    - [2.2.2.](#) POST
    - [2.2.3.](#) PUT
    - [2.2.4.](#) DELETE
    - [2.2.5.](#) SUBSCRIBE
  - [2.3.](#) URIs
  - [2.4.](#) CoAP Codes
  - [2.5.](#) Content-type encoding
- [3.](#) Message Formats
  - [3.1.](#) CoAP magic byte header
  - [3.2.](#) CoAP header
  - [3.3.](#) Header options
    - [3.3.1.](#) Content-type Option

<a href="#">3.3.2.</a>	Uri Option
<a href="#">3.3.3.</a>	Uri-code Option
<a href="#">3.3.4.</a>	Max-age Option
<a href="#">3.3.5.</a>	Etag Option
<a href="#">3.3.6.</a>	Date Option
<a href="#">3.3.7.</a>	Subscription-lifetime Option
<a href="#">4.</a>	Transport Binding
<a href="#">4.1.</a>	UDP
<a href="#">4.1.1.</a>	Retransmission
<a href="#">4.2.</a>	Datagram TLS
<a href="#">4.3.</a>	TCP
<a href="#">4.4.</a>	TLS
<a href="#">4.5.</a>	Default Port
<a href="#">5.</a>	Caching
<a href="#">5.1.</a>	Cache control
<a href="#">5.2.</a>	Cache refresh
<a href="#">5.3.</a>	Proxying
<a href="#">6.</a>	Resource Discovery
<a href="#">6.1.</a>	Link Format
<a href="#">7.</a>	HTTP Mapping
<a href="#">8.</a>	Protocol Constants
<a href="#">9.</a>	Examples
<a href="#">10.</a>	Security Considerations
<a href="#">11.</a>	IANA Considerations
<a href="#">11.1.</a>	Codes
<a href="#">11.2.</a>	Content Types
<a href="#">12.</a>	Acknowledgments
<a href="#">13.</a>	Changelog
<a href="#">14.</a>	References
<a href="#">14.1.</a>	Normative References
<a href="#">14.2.</a>	Informative References
<a href="#">§</a>	Authors' Addresses

---

## 1. Introduction

[TOC](#)

The use of web services on the Internet has become ubiquitous in most applications, and depends on the fundamental Representational State Transfer (REST) architecture of the web. The proposed Constrained RESTful Environments (CoRE) working group aims at realizing the REST architecture in a suitable form for the most constrained nodes (e.g. 8-bit microcontrollers with limited RAM and ROM) and networks (e.g. 6LoWPAN). One of the main goals of CoRE is to design a generic RESTful protocol for the special requirements of this constrained environment, especially considering energy and building automation applications.

This document specifies the RESTful Constrained Application Protocol (CoAP) which easily translates to HTTP for integration with the web while meeting specialized requirements such as multicast support, very low overhead and simplicity for constrained environments [\[I-D.shelby-core-coap-req\] \(Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R. Kelsey, "CoAP Requirements and Features," April 2010.\)](#). CoAP has the following main features:

- \*RESTful protocol design minimizing the complexity of mapping with HTTP.
- \*Low header overhead and parsing complexity.
- \*URI and Content-type support.
- \*Support for the discovery of resources provided by known CoAP end-points.
- \*Simple subscription for a resource and a resulting notification mechanism.
- \*Simple caching based on max-age.

The mapping of CoAP with HTTP is defined, allowing proxies to be built providing access to CoAP resources via HTTP in a uniform way.

---

## 2. Constrained Application Protocol

[TOC](#)

This section specifies the basic functionality and processing rules of CoAP.

---

### 2.1. Interaction Model

[TOC](#)

The interaction model of CoAP is client/server with request or notify messages initiating a transaction responded to with a matching response based on a transaction ID if the A bit is set. Machine-to-machine interactions with RESTful design typically result in a CoAP implementation acting in both client and server roles. A CoAP request is equivalent to an HTTP request, and is sent by a client to request an action (using a method) on a resource (identified by a URI) on a server. A CoAP notify is the inverse of a request, where a server sends a notify message to a client about a resource on the server (identified by a URI). A notify includes the representation, Etag and/or Date of

the resource. Examples message exchanges can be found from [Section 9 \(Examples\)](#).

This document specifies the interaction of two CoAP end-point acting as a client or server. We can consider an end-point of CoAP to be an application process using one source port for sending or receiving CoAP messages. Thus a host may run any number of CoAP end-points.

---

#### 2.1.1. Request messages

[TOC](#)

A CoAP end-point acting as a client sends a request with the following rules. The Version field is set to 0. The Type Flag is set to 0 indicating a request. For a UDP binding the A Flag SHOULD be set requesting a response and enabling retransmission in case of timeout (see [Section 4.1.1 \(Retransmission\)](#)). The A Flag MAY be unset for TCP bindings or for a UDP binding where reliability is too costly or not useful. The Method field MUST be set with a value of 0-4. The TRANSACTION\_ID variable is increased by one, and this value is placed in the Transaction ID Field. See [Section 2.1.4 \(Option fields\)](#) for options rules.

If a payload is to be included in the message, it immediately follows the last option or the Transaction ID if none. If a magic byte header is included, its Length value indicates the length of the message in octets. A magic byte header MUST be included if multiple messages are packed into a single UDP datagram or over TCP.

For each request sent with the A flag set, a CoAP end-point keeps track of the destination IP address and Transaction ID of the request for the purpose of matching responses. For unicast destination over UDP, the retransmission procedure is described in [Section 4.1.1 \(Retransmission\)](#).

Upon receiving a request, a CoAP end-point performs the following validation and processing:

- o The Version Field MUST be 0.
- o The Type Flag MUST be 0.
- o The Method Field MUST be 0-4.
- o If the Number of Options Field is > 0, then each option is validated and processed as in [Section 2.1.4 \(Option fields\)](#).
- o The length of the Payload is taken from the Content-length Option or calculated from the datagram length otherwise.
- o The Method, URI, any options and Payload are passed on to the corresponding application process.

- o If the A bit is set, an appropriate response message MUST be sent to the source IPv6 address and port of the request with the same Transaction ID of the request. If the A bit is unset, a response message MUST NOT be sent.

TODO: Define the behavior on failure. Error or silent discard?

---

### 2.1.2. Notify messages

[TOC](#)

The sending of a notify message is similar to sending a request message, with the following difference: The Type Flag is set to 2. The processing of a notify message is similar to processing a request message.

---

### 2.1.3. Response message

[TOC](#)

A response message is created with the following rules. The Version Field is set to 0. The Type Flag is set to 1. The Code is set to one of the supported response codes in [Section 11.1 \(Codes\)](#). The Transaction ID MUST be set to that of the corresponding request. See [Section 2.1.4 \(Option fields\)](#) for options rules. An optional Payload may be included as appropriate for the request.

Upon receiving a response, a CoAP end-point performs the following validation and processing:

- o The Version Field MUST be 0.
- o The Type Flag MUST be 1.
- o The Code Field MUST contain a valid code.
- o If the Number of Options Field is  $> 0$ , then each option is validated and processed as in [Section 2.1.4 \(Option fields\)](#).
- o The length of the Payload is taken from the Content-length Option or calculated from the datagram length otherwise.
- o The Transaction ID is used to match the response to an open request entry, and the response code, any options and Payload are passed on to the corresponding application process. If no match is found, the message is silently discarded.

TODO: Define the behavior on failure. Error or silent discard?

---

#### 2.1.4. Option fields

[TOC](#)

If no options are to be included, the Option Number Field is set to 0 and the Payload (if any) immediately follows the Transaction ID. If options are to be included, the following rules apply. The number of options is placed in the Number of Options Field. Each option is then placed in order of Type, immediately following the Transaction ID with no padding. Unknown options MUST be silently skipped.

TODO: Option validation and processing.

---

#### 2.1.5. Transaction IDs

[TOC](#)

The Transaction ID is a unique unsigned integer kept by a CoAP end-point for all of the CoAP request or notify messages it sends. Each CoAP end-point keeps a single Transaction ID variable, which is changed each time a new request or notify message is sent regardless of the destination address or port. The Transaction ID is used to match a response with an outstanding request or notify, for retransmission and to discard duplicate messages. The initial Transaction ID should be randomized.

---

### 2.2. Methods

[TOC](#)

CoAP supports the basic RESTful methods of GET, POST, PUT, DELETE, which are easily mapped to HTTP methods. In this section each method is defined along with its behavior. In addition, CoAP defines a new SUBSCRIBE method for requesting soft-state subscriptions for resources. As CoAP methods manipulate resources, they have the same properties of safe (only retrieval) and idempotent (you can invoke it multiple times with the same effects) as HTTP [Section 9.1 \(Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," June 1999.\)](#) [RFC2616]. The GET method is safe, therefore it SHOULD NOT take any other action on a resource other than retrieval. The GET, PUT and DELETE methods can be considered idempotent.

---

#### 2.2.1. GET

[TOC](#)

The GET method retrieves the information of the resource identified by the request URI. Upon success a 200 (OK) response SHOULD be sent.

All GET requests MUST be idempotent in that they produce no side-effects.

The response to a GET is cacheable if it meets the requirements in [Section 5 \(Caching\)](#).

---

#### **2.2.2. POST**

[TOC](#)

The POST method is used to request the server to create a new resource under the requested URI. If a resource has been created on the server, the response should be 201 (Created) including the URI of the new resource in the header and any possible status in the message body. If the POST does not result in a new resource being created on the server, a 200 (OK) response code is returned.

Responses to this method are not cacheable.

---

#### **2.2.3. PUT**

[TOC](#)

The PUT method requests that the resource identified by the request URI be updated with the enclosed message body. If a resource exists at that URI the message body SHOULD be considered a modified version of that resource. If no resource exists then the server MAY create a new resource with that URI.

All PUT requests MUST be idempotent in that they produce no side-effects.

Responses to this method are not cacheable.

---

#### **2.2.4. DELETE**

[TOC](#)

The DELETE method requests that the resource identified by the request URI be deleted. The response 200 (OK) SHOULD be sent on success.

All DELETE requests MUST be idempotent in that they produce no side-effects.

Responses to this method are not cacheable.

---

#### **2.2.5. SUBSCRIBE**

[TOC](#)

CoAP supports a built-in subscribe/notify push model for an end-point to notify another end-point about a resource of interest. This push is accomplished using the CoAP notify message type, whose URI corresponds



to the resource of interest on the end-point sending the notify message. A notify may include the latest representation of the resource in its payload and/or the Etag Option.

The SUBSCRIBE method allows an end-point to request notifications about a resource. A request of this method MAY include the Subscription-lifetime Option defined in [Section 3.3.7 \(Subscription-lifetime Option\)](#). In the absence of this Option, its maximum lifetime is assumed. End-points MUST NOT send notify messages without a valid subscription. Subscriptions are soft-state, and must be refreshed by sending a new SUBSCRIBE message before the end of its lifetime. Servers keep track of subscriptions its resources, and upon change a notify message is sent to the source address and port of the original SUBSCRIBE request with the URI of the resource in question. Notifications MAY be sent with the A bit set, enabling a server to detect if a subscriber is no longer valid. A subscription SHOULD be removed after MAX\_RETRANSMIT failures when the A bit is set. A server is not required to support subscriptions for its resources, and MAY limit the number of simultaneous subscriptions.

---

### 2.3. URIs

[TOC](#)

The Universal Resource Identifier (URI) [\[RFC3986\] \(Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier \(URI\): Generic Syntax," January 2005.\)](#) is an important feature of the REST architecture, where the relative part of the URI indicates which resource is being manipulated. CoAP supports variable-length string URIs with the Uri Option. As this URI is used as a locator, CoAP only supports Universal Resource Locator features of [\[RFC3986\] \(Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier \(URI\): Generic Syntax," January 2005.\)](#) although throughout the document we refer to URI. CoAP supports relative references in the Uri Option (e.g. /sensors/temperature) for messages to a CoAP end-point, and absolute URIs for use with a proxy (coap://2001:1ba3::450a/sensors/temperature), and does not support "." and ".." schemes. A CoAP implementation MAY support query "?" processing if needed, however fragment "#" processing is not supported. IRIs are not supported. All URI strings in CoAP MUST use the US-ASCII encoding defined in [\[RFC3986\] \(Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier \(URI\): Generic Syntax," January 2005.\)](#).

The CoAP protocol scheme is identified in URIs with "coap://" (TODO: IANA considerations).

All resources URIs MUST begin with a leading "/" slash character.

During transmission of the URI in the Uri header, the leading slash MAY be omitted. If the Uri header does start with a "/" leading slash, then it is implied.

TODO: Describe the use of binary URI codes and the Uri-code Option.

---

## 2.4. CoAP Codes

[TOC](#)

When a response message is sent in response to a request or notify message it MUST always include a response code in the header. Notify messages also include a code field, which is set to "200 OK" by default. CoAP makes use of a subset of HTTP response codes as defined in [Section 11.1 \(Codes\)](#).

---

## 2.5. Content-type encoding

[TOC](#)

In order to support heterogeneous uses, CoAP is transparent to the use of different application payloads. In order for the application process receiving a packet to properly parse a payload, its content-type should be explicitly known from the header (as e.g. with HTTP). The use of typical binary encodings for XML is discussed in [\[I-D.shelby-6lowapp-encoding\] \(Shelby, Z., Luimula, M., and D. Peintner, "Efficient XML Encoding and 6LowApp," October 2009.\)](#). It is obvious that string names of Internet media types [\[RFC2046\] \(Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions \(MIME\) Part Two: Media Types," November 1996.\)](#) are not appropriate for use in the CoAP header. Instead, CoAP simply assigns identifiers to a subset of common MIME and content transfer encoding types. The content-type identifier is optionally included in the Content-type Option Header of messages to indicate the type of the message body. CoAP Content-type identifiers are defined in [Section 11.2 \(Content Types\)](#).

---

## 3. Message Formats

[TOC](#)

CoAP makes use of three message types - request, notify and response, using a simple binary header format. This base header may be followed by options in Type-Length-Value (TLV) format. CoAP is by default bound to UDP and optionally to TCP as described in [Section 4 \(Transport Binding\)](#). In addition, a CoAP magic byte header is used when multiple messages are packed into a UDP payload and over TCP.

Any bytes after the headers in the packet are considered the message payload, if any. The length of the message payload is implied by the datagram length or the Length Field of the magic byte header if included. When bound to UDP the entire message MUST fit within a single datagram. When used with 6LoWPAN [\[RFC4944\] \(Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets](#)

over [IEEE 802.15.4 Networks," September 2007.](#)), messages SHOULD fit into a single 802.15.4 frame to minimize fragmentation.

---

### 3.1. CoAP magic byte header

[TOC](#)

This section defines the CoAP magic byte header shown in [Figure 1 \(CoAP magic byte header\)](#). This header provides a clear delimiter between CoAP messages and the total message length. The magic byte header MUST be included before the CoAP header when packing multiple messages in a single UDP datagram or over a TCP connection, and MAY be included otherwise.

---

Template:

```
0
0 1 2 3 4 5 6 7 8
+---+---+---+---+---+---+
|           'r'           |X|
+---+---+---+---+---+---+
```

Length of 0-127:

```
0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           'r'           |0|   Length   |           CoAP header ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

Length of 128-32768:

```
0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
|           'r'           |1|           Length           | CoAP header ...
+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+---+
```

---

**Figure 1: CoAP magic byte header**

---

**'r':**

CoAP magic byte which is US-ASCII character 'r' = 0x72 = 0b01110010. The magic byte 'r' stands for RESTful :-)

**X:** 1-bit Extended Length Flag. When 0 the Length is a 7-bit unsigned integer. When 1 the Length Field is extended by an octet and Length Field is a 15-bit unsigned integer.

**Len:** Length Field. When X is 0 Length is a 7-bit unsigned integer allowing values of 0-127 octets. When X is 1 Length is a 15-bit unsigned integer allowing values of 128-32767 octets. The Length field indicates the length of the CoAP message following the Length Field in octets.

**CoAP header** The CoAP header immediately follows the Length Field.

---

### 3.2. CoAP header

[TOC](#)

This section defines the CoAP header, which is shared for all message types.

**Request:** A CoAP request message is sent by a client to request a URI on a server using one of the methods listed in [Table 1 \(Method codes\)](#).

**Notify:** A CoAP notify message is sent by a server to notify a client about a resource (identified by a URI) on the server as a result of a valid subscription for that resource.

**Response:** A CoAP response message is sent in response to a CoAP request or notify when appropriate. Responses include a Transaction ID corresponding to that of the request. A response is always sent when the A flag is set in a request, and is never sent when the A flag is not set. A response is always sent to the source IP address and port of the corresponding request or notify.

---

Template:

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
|Ver| T |    0    | Type Specific |           Transaction ID           |
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
|  Options (if any) ...
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
|  Payload (if any) ...
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+

```

Request ( $T=0$ ):

```

0                                     1                                     2                                     3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
|Ver|0 0|    0    |A|  R  | Meth  |          Transaction ID          |
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
|  Options (if any) ...
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
|  Payload (if any) ...
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+

```

Response (T=1):

```

0                               1                               2                               3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
|Ver|0 1|    0   | R |      Code     |           Transaction ID          |
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
|  Options (if any) ...
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
|  Payload (if any) ...
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+

```

Notify ( $T=2$ ):

```

0          1          2          3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
|Ver|1 0|    0   |A|R|      Code   |           Transaction ID           |
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
|  Options (if any) ...
+-+-+-----+-----+-----+-----+-----+-----+-----+-----+
|  Payload (if any) ...

```



---

Method	Code
GET	0
POST	1
PUT	2
DELETE	3
SUBSCRIBE	4

**Table 1: Method codes**

---

### 3.3. Header options

[TOC](#)

CoAP messages may also include one or more header options in TLV format. Each option has the following format:

---

Template:

```
0
0 1 2 3 4 5
+--+--+--+--+
|  Type  |X|
+--+--+--+--+
```

Length of 0-4:

```
0                                1                                2                                3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  Type  |0|Len|  Option Value ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

Length of 5-1024:

```
0                                1                                2                                3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
|  Type  |1|          Len          |  Option Value ...
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+
```

**Figure 3: Header option format**

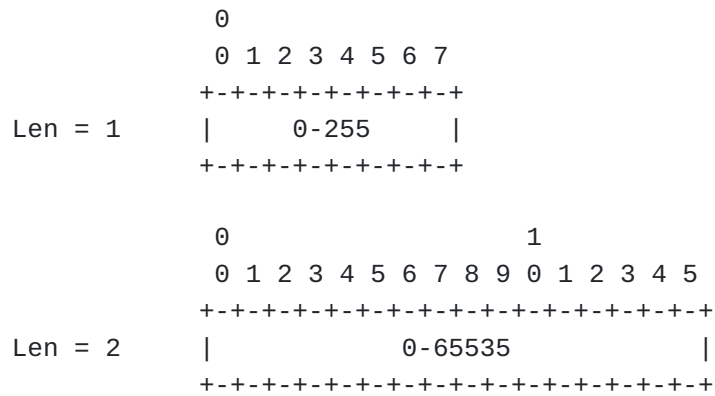
**Type:** 5-bit unsigned integer. The type of the option as defined in [Table 2 \(Option headers\)](#), allowing for up to 32 options. Future specifications may define new CoAP option types. Option types 30-32 are reserved for experimental purposes.

**X:** 1-bit Extended Length Flag. When 0 the Length is a 2-bit unsigned integer. When 1 the option header is extended by an octet and Length is a 10-bit unsigned integer.

**Len:** Length Field. When X is 0 Length is a 2-bit unsigned integer allowing values of 0-4 octets. When X is 1 Length is a 10-bit unsigned integer allowing values of 5-1024 octets.

**Option Value** The value in the format defined for that option in [Table 2 \(Option headers\)](#) with a length of Option Len octets. Options may use variable length unsigned integer values of Len Field octets in network byte order as shown in [Figure 4 \(Variable length unsigned integer value format\)](#).





Len = 3 is 24 bits, Len = 4 is 32 bits etc.

**Figure 4: Variable length unsigned integer value format**

The following options are defined in this document.

Type	Name	Data type	Length	Rules
0	Content-type	Variable uint	1-2 B	
1	Uri	String	1-32768 B	Never in response
2	Uri-code	Variable uint	1-2 B	Never with Uri Option
3	Max-age	Variable uint	1-4 B	
4	Etag	Variable uint	1-4 B	
5	Date	Variable integer	4-6 B	Never in request
6	Subscription-lifetime	Variable uint	1-3 B	With SUBSCRIBE or its response

**Table 2: Option headers**

### 3.3.1. Content-type Option

The Content-type Identifier Option indicates the Internet Media Type of the message-body, see [Section 11.2 \(Content Types\)](#) for the encoding and identifier tables. A Content-type Identifier Option SHOULD be included if there is a payload included with a CoAP message, and MUST not be included for a zero-length payload.

---

### 3.3.2. Uri Option

[TOC](#)

The Uri Option indicates the string URI of the resource that may be included in request and notify messages. In the absence of this option, the URI is assumed to be "/". [Section 2.3 \(URIs\)](#) specifies the rules for URIs in CoAP.

---

### 3.3.3. Uri-code Option

[TOC](#)

The Uri-code Option is used as an alternative to the Uri Option, and indicates a variable length code assigned globally to well-known URI strings or by the CoAP end-point in a link description entry "code" field (see [Section 6 \(Resource Discovery\)](#)). The Uri-code Option MUST NOT be used at the same time as the Uri Option. If both options appear in a message then the Uri-code Option is ignored.

---

### 3.3.4. Max-age Option

[TOC](#)

The Max-age Option indicates the maximum age of the resource for use in cache control in seconds. The option is represented as a variable length unsigned integer maximum 32-bits in length. When included in a request, Max-age indicates the maximum age of a cached representation of that resource the client will accept. When included in a response or a notify, Max-age indicates the maximum time the representation may be cached before it MUST be discarded.

---

### 3.3.5. Etag Option

[TOC](#)

The Etag Option is a variable length unsigned integer which specifies the version of a resource representation. An Etag may be generated for a resource in any number of ways including a version, checksum, hash or

time. An end-point receiving an Etag MUST treat it as opaque and make no assumptions about its format. The Etag MAY be included in a notify message to indicate to a client if a resource has changed.

---

### 3.3.6. Date Option

[TOC](#)

The Date Option indicates the creation time and date of a given resource representation. It MAY be used in response and notify messages. The integer value is the number of seconds, excluding leap seconds, after midnight UTC, January 1, 1970. This time format cannot represent time values prior to January 1, 1970. The latest UTC time value that can be represented by a 31 bit integer value is 03:14:07 on January 19, 2038. Time values beyond 03:14:07 on January 19, 2038, are represented by 39 bit integer values which is sufficient to represent dates that should be enough for anyone. For applications requiring more accuracy, a 48-bit integer MAY be included representing this value in milliseconds instead of seconds.

---

### 3.3.7. Subscription-lifetime Option

[TOC](#)

The Subscription-lifetime Option indicates the subscription lifetime and is optionally included with the SUBSCRIBE method (see [Section 2.2.5 \(SUBSCRIBE\)](#)). The corresponding response MUST include a Subscription-lifetime Option confirming (or modifying) the subscription lifetime. The value of this option is a variable length unsigned integer up to 24-bits indicating the lifetime of the subscription in seconds with a maximum value of 194 days. In a response the server MAY return a different value that fits its own scheduling better. A value of all 0 in a request indicates cancellation of a subscription and in a response indicates subscription failure or rejection.

---

## 4. Transport Binding

[TOC](#)

The CoAP protocol will operate by default over UDP. There may be optional functions in CoAP (e.g. delivery of larger chunks of data) which if implemented are implemented over TCP. This section defines the binding of CoAP over UDP and TCP.

---

[TOC](#)

## 4.1. UDP

The goal of binding CoAP to UDP is to provide the bare minimum features for the protocol to operate over UDP, without trying to re-create the full feature set of TCP. CoAP over UDP has the following features:

- \*Simple stop-and-wait retransmission reliability with exponential back-off as described in [Section 4.1.1 \(Retransmission\)](#) when the A Flag is set.
- \*Transaction ID for response matching as described in [Section 2.1.5 \(Transaction IDs\)](#).
- \*Multicast support without retransmission. CoAP supports the use of multicast destination addresses when bound to UDP. Although the A bit may be used to force a response, retransmission MUST NOT be performed.

When a single CoAP message is sent using UDP, the length of the Payload can be calculated from the datagram length. Multiple CoAP messages MAY also be concatenated in a single UDP payload. In this case each message header MUST be precluded by a magic byte header making the start and length of each message explicit. When multiple messages are packed in a UDP payload, they are processed by the CoAP end-point in order and independently. The responses to packed messages SHOULD also be packed if space permits.

When bound to UDP the entire message MUST fit within a single datagram of length 576 octets over IPv4 and 1280 octets over IPv6. When used with 6LoWPAN [\[RFC4944\] \(Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," September 2007.\)](#), messages SHOULD fit into a single link-layer frame to minimize fragmentation (often on the order of 60-90 octets).

---

### 4.1.1. Retransmission

[TOC](#)

A CoAP end-point keeps track of open request or notify messages expecting a response (A Flag set) using a conceptual data structure or entries awaiting a response. Each entry includes at least the destination address and port of the original message, the message itself, a retransmission counter (UDP only) and a timeout. When a request or notify message is sent with the A Flag set, an entry is made for that message with a default initial timeout of RESPONSE\_TIMEOUT and the retransmission counter set to 0. When a matching response is received for an entry, the entry is removed. When a timeout is triggered for an entry and the retransmission counter is less than MAX\_RETRANSMIT, the original message is retransmitted to the destination without modification, the retransmission counter is

incremented, and the timeout is doubled. If the retransmission counter reaches MAX\_RETRANSMIT on a timeout, then the entry is removed and the application process informed of delivery failure. For CoAP messages sent to IP multicast addresses, retransmission MUST NOT be performed. Therefore MAX\_RETRANSMIT is always set to 0 when the destination address is multicast.

---

## 4.2. Datagram TLS

[TOC](#)

CoAP may also be bound to Datagram Transport Layer Security [\[RFC4347\]](#) (Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security," April 2006.) in order to prevent eavesdropping, tampering, or message forgery.

TODO: Define the DTLS binding in detail. Expected as a contribution from security people in the WG. Use the simplest possible configuration and AES-128 encryption as this is usually supported by e.g. IEEE 802.15.4 hardware. The current suggestion is to make AES, RSA, and SHA1 mandatory to implement and one with SHA256 RECOMMENDED.

---

## 4.3. TCP

[TOC](#)

The CoAP protocol also may also be bound to TCP in special cases. As TCP provides a reliable stream this binding does not require anything special from the CoAP protocol design. Retransmission MUST BE disabled, thus MAX\_RETRANSMIT is always set to 0. The Transaction ID MUST also be used over TCP and the magic byte header MUST always be included. CoAP end-points are not expected to support both UDP and TCP, and SHOULD NOT attempt to dynamically negotiate between transports.

The following cases have been identified where TCP MAY be used:

- \*When an application returns large resource representations, which do not fit in a single datagram.

- \*For providing congestion control if CoAP is being applied across the wider Internet in a topology where congestion is a concern.

It should be noted that using a similar configuration. CoAP could be used over other stream media such as serial ports. Such a configuration is however out of scope of this document.

---

[TOC](#)

#### 4.4. TLS

CoAP may also be bound to Transport Layer Security [\[RFC4346\] \(Dierks, T. and E. Rescorla, "The Transport Layer Security \(TLS\) Protocol Version 1.1," April 2006.\)](#) in order to prevent eavesdropping, tampering, or message forgery.

TODO: Define the TLS binding in detail. Expected as a contribution from security people in the WG. Use the simplest possible configuration and AES-128 encryption as this is usually supported by e.g. IEEE 802.15.4 hardware. The current suggestion is to make AES, RSA, and SHA1 mandatory to implement and one with SHA256 RECOMMENDED.

---

#### 4.5. Default Port

[TOC](#)

CoAP has a default port of 61616 (TODO: IANA) which is within the compressed UDP port space defined in [\[RFC4944\] \(Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, "Transmission of IPv6 Packets over IEEE 802.15.4 Networks," September 2007.\)](#). This default port is the same for UDP and TCP.

---

### 5. Caching

[TOC](#)

CoAP end-points are by definition constrained by bandwidth and processing power. To optimize the performance of data transfer under these constraints, we leverage caching features consistent with HTTP. Caching includes the following concepts:

- \*cache life of a resource is controlled via the Max-Age header option
  - \*cache refresh and versioning of a resource is controlled via the Etag header option
  - \*proxies between a client and end-point may participate in the caching process on behalf of sleeping end-points and to avoid unnecessary traffic on the constrained network
- 

#### 5.1. Cache control

[TOC](#)

When an end-point publishes a resource for a GET request, it SHOULD specify the Max-Age header option. The Max-Age specifies the cache life

of the resource in seconds. Resources which change rapidly will have a short cache life, and resources which change infrequently should specify a long cache life. If Max-Age is unspecified in a GET response, then it is assumed to be 60 seconds. If an end-point wishes to disable caching, it must explicitly specify a Max-Age of zero seconds.

When a client reads the response from a GET request, it should cache the resource representation for the cache lifetime as specified by the Max-Age header. During the cache lifetime, the client SHOULD use its cached version and avoid performing additional GETs for the resource. In general, the origin server end-point is responsible for determining cache age. However, in some cases a client may wish to determine its own tolerance for cache staleness. In this case, a client may specify the Max-Age header during a GET request. If the client's Max-Age is of a shorter duration than the age of a cached resource, then the proxy or end-point SHOULD perform a cache refresh. If the client specifies a Max-Age of zero seconds, then the response MUST discard the cached representation and return a fresh representation.

---

## 5.2. Cache refresh

[TOC](#)

After the expiration of the cache lifetime, clients and proxies must refresh their cached representation of a resource. Cache refresh is accomplished using GET request which will return a representation of the resource's current state.

If the end-point has the capability to version the resource, then the end-point should include the Etag header option in the response to a GET request. The Etag is a variable length integer which captures a version checksum of the resource. The Etag is an opaque identifier; clients MUST NOT infer any semantics from the Etag value.

If an end-point specifies the Etag header option, then the client SHOULD specify a matching Etag header option in their GET request during cache refresh. If the end-point's version of the resource is unmodified, then it SHOULD specify a 304 response with no payload to avoid retransmitting the resource representation.

---

## 5.3. Proxying

[TOC](#)

See [\[I-D.frank-6lowapp-chopan\]](#) (Frank, B., "Chopan - Compressed HTTP Over PANs," September 2009.).

TODO:

- \*Are interception proxies are still required to deal with a) sleeping nodes and b) protecting Internet HTTP traffic from overwhelming the CoAP network?

\*But interception proxies breaks end-to-end IP encapsulation and requires support at the routing level

\*Often the interception proxy is the same as the HTTP-to-CoAP gateway, so we need to decide how these topics dovetail

\*In Chopan, the sleeping problem was tackled by having sleeping nodes check-in with their proxies while awake, notify model might solve this problem to some extent but still have to coordinate the sleep/awake times

\*In Chopan we actually used caching to deal with POSTs, etc because otherwise how do you send a request to a sleeping node? The current caching sections are to be exclusive to GETs, but we still need to solve the problem for other types if methods.

---

## 6. Resource Discovery

[TOC](#)

The discovery of resources offered by a CoAP end-point is extremely important in machine-to-machine applications where there are no humans in the loop and static interfaces results in fragility. The discovery of resources provided by an HTTP Web Server is called Web Discovery. In this document we refer to the discovery of resources offered by a CoAP end-point as Resource Discovery.

CoAP makes the assumption that end-points are available on the default CoAP port, or otherwise have been configured or discovered using some general service discovery mechanism such as

[\[I-D.cheshire-dnsext-multicastdns\]](#) (Cheshire, S. and M. Krochmal, "Multicast DNS," March 2010.). This section assumes that such a configuration or service discovery has already been performed, if needed.

Resource Discovery in CoAP is accomplished through the use of well-known resources which describe the links offered by that CoAP end-point. Well-known resources have the URI form `"/.well-known/"` as specified in [\[RFC5785\]](#) (Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)," April 2010.). Every CoAP end-point MUST support this well-known resource. The resource representation of this is described in [Section 6.1 \(Link Format\)](#). CoAP requests the following well-known URL for discovery: `"/.well-known/resources"` (TODO: Formal description for use in request as per [\[RFC5785\]](#) (Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)," April 2010.)).

CoAP Resource Discovery supports the following interactions:

\*[request GET `/.well-known/resources`] returns the list of links available from a CoAP end-point.



\*A CoAP end-point may notify interested clients when this description has changed by sending [notify /.well-known/resources]. This resources MAY support subscription.

\*More capable end-points such as proxies MAY support a resource directory by accepting [request POST /.well-known/resources] messages from other CoAP end-points. This adds the resources of other end-points to an agent directory in which absolute URIs are included for the links.

End-points with a large number of resources SHOULD organize their resource descriptions into a hierarchy of link resources. This is done by including links in the /.well-known/resources list which point to other resource lists, e.g. /.well-known/resources/sensors.

---

## 6.1. Link Format

[TOC](#)

CoAP resource discovery makes use of the HTTP Link Header format specified in [\[I-D.nottingham-http-link-header\] \(Nottingham, M., "Web Linking," April 2010.\)](#). This specification allows for the use of this simple link format by other protocols, thus not limiting it to the actual HTTP Link Header. The format does not require special XML or binary parsing, and is extensible.

CoAP defines a subset of the [\[I-D.nottingham-http-link-header\] \(Nottingham, M., "Web Linking," April 2010.\)](#) features and specific parameters that have known meaning for CoAP resource discovery. A CoAP end-point MAY make use of link extension parameters as needed. The CoAP link format does not start with the "Link:" text. The following formal description is used for forming and parsing this link format:

```

link-value      = "<" URI-Reference ">" *( ";" link-param )
link-param      = ( ( "rel" "=" URI )
                    | ( "name" "=" quoted-string )
                    | ( "type" "=" ( media-type | media-code ) )
                    | ( "id" "=" integer )
                    | ( "code" "=" integer )
                    | ( link-extension ) )
link-extension  = ( parmname [ "=" ( ptoken | quoted-string ) ] )
ptoken          = 1*ptokenchar
ptokenchar      = "!" | "#" | "$" | "%" | "&" | "'" | "("
                  | ")" | "*" | "+" | "-" | "." | "/" | DIGIT
                  | ":" | "<" | "=" | ">" | "?" | "@" | ALPHA
                  | "[" | "]" | "^" | "_" | "`" | "{" | "|"
                  | "}" | "~"
media-code      = see Appendix B
media-type      = type-name "/" subtype-name

```

The link-value is the relative URI of the resource on that end-point or an absolute URI in the case of a directory agent. The rel parameter is a URI that points to the definition of that resource interface, for example in WADL. The name parameter is a descriptive or ontology name of the resource class. This name parameter SHOULD be in an m-DNS [\[I-D.cheshire-dnsext-multicastdns\] \(Cheshire, S. and M. Krochmal, "Multicast DNS," March 2010.\)](#) compatible form. The type parameter includes Internet media type this resource returns in ascii or code format. The id field is a unique identifier (e.g. UUID) for this resource for use in e.g. search directories. Finally, the code field is used to identify this resource for reference with the Uri-code Option. All link parameters are optional and custom link-extensions may be defined. An example of a typical CoAP link description in this format would be:

```

</sensor/temp>; rel="sensor.wadl"; name="TemperatureC"; type=text/plain
</sensor/light>; rel="sensor.wadl"; name="LightLux"; type=text/plain

```

---

## 7. HTTP Mapping

[TOC](#)

TODO.

---

## 8. Protocol Constants

[TOC](#)

This section defines the relevant protocol constants defined in this document:

**RESPONSE\_TIMEOUT** 0.5 seconds

**MAX\_RETRANSMIT** 5

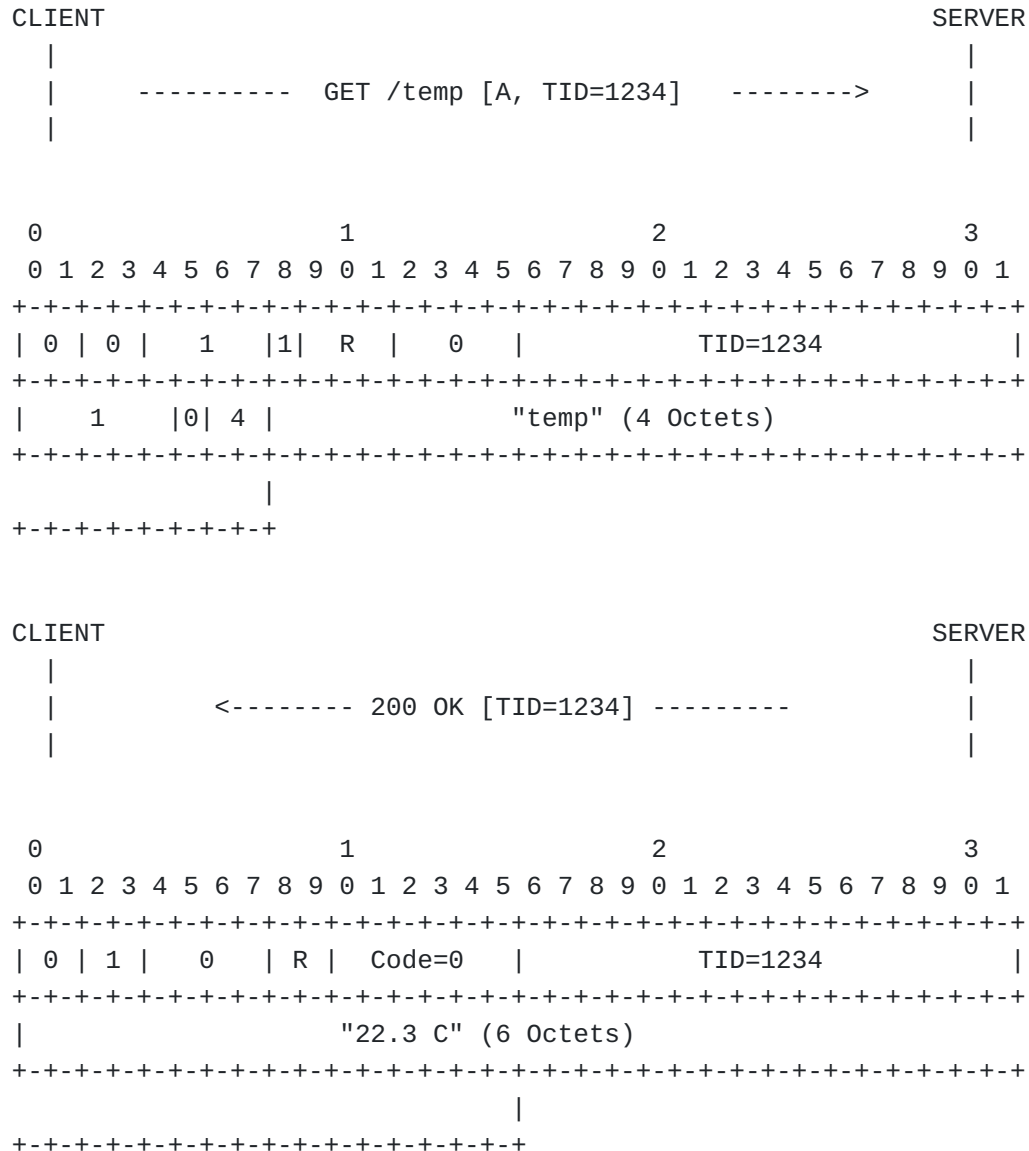
---

## 9. Examples

[TOC](#)

[Figure 5 \(Basic request/response\)](#) shows a basic request/response sequence. A client makes a GET request for the resource /temp to the server. The A Flag is set, requesting a response and the Transaction ID is 1234. The request includes one Uri Option "temp" of Len = 4. This request is 9 octets long. The corresponding response is of code 200 OK and includes a text/plain Payload of "22.3 C". The Transaction ID is 1234, thus the transaction is successfully completed. The response is 10 octets long.

---



**Figure 5: Basic request/response**

TODO: Request with multiple packed messages (magic byte example..)  
 TODO: Request - Response (with retransmission)  
 TODO: Request - Response (discovery)  
 TODO: Request - Response (with subscription)... Resulting Notify - Response

## 10. Security Considerations

TODO: Expand this section to a full security analysis, including how to use CoAP with various security options.

Some of the features considered in this document will need further security considerations during a protocol design. For example the use of string URLs may have entail security risks due to complex processing on limited microcontroller implementations.

The CoAP protocol will be designed for use with e.g. (D)TLS or object security. A protocol design should consider how integration with these security methods will be done, how to secure the CoAP header and other implications.

---

## 11. IANA Considerations

[TOC](#)

TODO (See IANA comments in the document).

---

### 11.1. Codes

[TOC](#)

CoAP makes use of (a subset of) the HTTP status codes defined in [\[RFC2616\]](#) (Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1," June 1999.). The HTTP status code is encoded into a 6-bit unsigned integer code with the mapping defined in [Table 3 \(CoAP Codes\)](#). The use of these codes is defined throughout this document using the HTTP Name.

---

Code	HTTP Name
0	200 OK
1	201 Created
14	304 Not Modified
20	400 Bad Request
21	401 Unauthorized
23	403 Forbidden
24	404 Not Found
25	405 Method Not Allowed
29	409 Conflict
35	415 Unsupported Media Type

40	500 Internal Server Error
43	503 Service Unavailable
44	504 Gateway Timeout

**Table 3: CoAP Codes**

## 11.2. Content Types

[TOC](#)

Internet media types are identified by a string in HTTP, such as "application/xml". This string is made up of a top-level type "application" and a sub-type "xml" [\[RFC2046\] \(Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions \(MIME\) Part Two: Media Types," November 1996.\)](#). In order to minimize the overhead of using these media types to indicate the type of message payload, CoAP defines an identifier encoding scheme for a subset of Internet media types. It is expected that this table of identifiers will be extensible and maintained by IANA.

The Content-type Option is formatted as a variable length unsigned integer, thus the most common media types are encoded into an 8-bit unsigned integer. This identifier is encoded as follows. Regardless of the length of the integer, the most significant 3 bits indicates the top-level media type (text, application etc.) as defined in [Table 4 \(Top-level type identifiers\)](#). The five initial top-level types defined in [\[RFC2046\] \(Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions \(MIME\) Part Two: Media Types," November 1996.\)](#) are supported. Composite high-level types (multipart and message) are not supported. The remaining bits indicate the sub-types [\[RFC2046\] \(Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions \(MIME\) Part Two: Media Types," November 1996.\)](#). This allows for up to 8 high-level types, with up to 32 sub-types for each in an 8-bit identifier and up to 8192 sub-types in a 16-bit identifier. For example, "application/xml" would be encoded in 8-bits as:

5 << 5 | 00 = 10100000

Top-level type	Identifier
text	1

image	2
audio	3
video	4
application	5

**Table 4: Top-level type identifiers**

Sub-type	Identifier
xml	0
plain	1
csv	2
html	3

**Table 5: text sub-type identifiers**

Sub-type	Identifier
gif	0
jpeg	1
png	2
tiff	3

**Table 6: image sub-type identifiers**

Sub-type	Identifier
raw	0

**Table 7: audio sub-type identifiers**

---

Sub-type	Identifier
raw	0

**Table 8: video sub-type identifiers**

---

Sub-type	Identifier
xml	0
octet-stream	1
rdf+xml	2
soap+xml	3
atom+xml	4
xmpp+xml	5
exi	6
x-bxml	7
fastinfoset	8
soap+fastinfoset	9
json	10

**Table 9: application sub-type identifiers**

---

## 12. Acknowledgments

[TOC](#)

Thanks to Carsten Bormann, Michael Stuber, Richard Kelsey, Cullen Jennings, Guido Moritz, Peter Van Der Stok, Adriano Pezzuto, Lisa Dussealt, Alexey Melnikov, Gilbert Clark, Salvatore Loreto, Petri Mutka, Szymon Sasin, Robert Quattlebaum, Robert Cragie, Angelo Castellani and David Ryan for helpful comments and discussions.

---

[TOC](#)



## 13. Changelog

Changes from -00 to -01:

- o Unified the message header and added a notify message type.
- o Renamed methods with HTTP names and removed the NOTIFY method.
- o Added a number of options field to the header.
- o Combines the Option Type and Length into an 8-bit field.
- o Added the magic byte header.
- o Added new Etag option.
- o Added new Date option.
- o Added new Subscription option.
- o Completed the HTTP Code - CoAP Code mapping table appendix.
- o Completed the Content-type Identifier appendix and tables.
- o Added more simplifications for URI support.
- o Initial subscription and discovery sections.
- o A Flag requirements simplified.

---

## 14. References

[TOC](#)

---

### 14.1. Normative References

[TOC](#)

[I-D.frank-6lowapp-chopan]	Frank, B., " <a href="#">Chopan - Compressed HTTP Over PANs</a> ," draft-frank-6lowapp-chopan-00 (work in progress), September 2009 ( <a href="#">TXT</a> ).
[I-D.nottingham-http-link-header]	Nottingham, M., " <a href="#">Web Linking</a> ," draft-nottingham-http-link-header-09 (work in progress), April 2010 ( <a href="#">TXT</a> ).
[RFC2046]	<a href="#">Freed, N.</a> and <a href="#">N. Borenstein</a> , " <a href="#">Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types</a> ," RFC 2046, November 1996 ( <a href="#">TXT</a> ).
[RFC2616]	

	<a href="#">Fielding, R.</a> , <a href="#">Gettys, J.</a> , <a href="#">Mogul, J.</a> , <a href="#">Frystyk, H.</a> , <a href="#">Masinter, L.</a> , <a href="#">Leach, P.</a> , and <a href="#">T. Berners-Lee</a> , " <a href="#">Hypertext Transfer Protocol -- HTTP/1.1</a> ," RFC 2616, June 1999 ( <a href="#">TXT</a> , <a href="#">PS</a> , <a href="#">PDF</a> , <a href="#">HTML</a> , <a href="#">XML</a> ).
[RFC3986]	<a href="#">Berners-Lee, T.</a> , <a href="#">Fielding, R.</a> , and <a href="#">L. Masinter</a> , " <a href="#">Uniform Resource Identifier (URI): Generic Syntax</a> ," STD 66, RFC 3986, January 2005 ( <a href="#">TXT</a> , <a href="#">HTML</a> , <a href="#">XML</a> ).
[RFC4346]	Dierks, T. and E. Rescorla, " <a href="#">The Transport Layer Security (TLS) Protocol Version 1.1</a> ," RFC 4346, April 2006 ( <a href="#">TXT</a> ).
[RFC4347]	Rescorla, E. and N. Modadugu, " <a href="#">Datagram Transport Layer Security</a> ," RFC 4347, April 2006 ( <a href="#">TXT</a> ).
[RFC5785]	Nottingham, M. and E. Hammer-Lahav, " <a href="#">Defining Well-Known Uniform Resource Identifiers (URIs)</a> ," RFC 5785, April 2010 ( <a href="#">TXT</a> ).

---

## 14.2. Informative References

[TOC](#)

[I-D.cheshire-dnsext-multicastdns]	Cheshire, S. and M. Krochmal, " <a href="#">Multicast DNS</a> ," draft-cheshire-dnsext-multicastdns-11 (work in progress), March 2010 ( <a href="#">TXT</a> ).
[I-D.shelby-6lowapp-encoding]	Shelby, Z., Luimula, M., and D. Peintner, " <a href="#">Efficient XML Encoding and 6LowApp</a> ," draft-shelby-6lowapp-encoding-00 (work in progress), October 2009 ( <a href="#">TXT</a> ).
[I-D.shelby-core-coap-req]	Shelby, Z., Stuber, M., Sturek, D., Frank, B., and R. Kelsey, " <a href="#">CoAP Requirements and Features</a> ," draft-shelby-core-coap-req-01 (work in progress), April 2010 ( <a href="#">TXT</a> ).
[RFC4944]	Montenegro, G., Kushalnagar, N., Hui, J., and D. Culler, " <a href="#">Transmission of IPv6 Packets over IEEE 802.15.4 Networks</a> ," RFC 4944, September 2007 ( <a href="#">TXT</a> ).

---

## Authors' Addresses

[TOC](#)

	Zach Shelby
	Sensinode
	Kidekuja 2
	Vuokatti 88600
	FINLAND
Phone:	+358407796297
Email:	<a href="mailto:zach@sensinode.com">zach@sensinode.com</a>

	Brian Frank
	SkyFoundry
	Richmond, VA
	USA
Phone:	
Email:	<a href="mailto:brian@skyfoundry.com">brian@skyfoundry.com</a>
	Don Sturek
	Pacific Gas & Electric
	77 Beale Street
	San Francisco, CA
	USA
Phone:	+1-619-504-3615
Email:	<a href="mailto:d.sturek@att.net">d.sturek@att.net</a>