core
Internet-Draft
Intended status: Standards Track
Expires: June 13, 2014

R. Mietz
University of Luebeck

December 10, 2013

# CoAP High-Level State Option Extension
# draft-mietz-core-coap-state-option-01

Abstract

CoAP is a RESTful application protocol for constrained devices which are often equipped with sensors measuring a physical phenomenon such as temperature on a precise scale. These sensor values are made available by a resource on the CoAP endpoint. However, for many applications it is not necessary to have the full precision a sensor can provide. It's often even enough to only have some high-level states instead of raw values. This document presents a new option for CoAP to dynamically create new resources for a sensor which provides user-defined high-level states instead of raw sensor values.

Table of Contents

1. Introduction . . . . . 3
1.1. Motivation . . . . . 3
1.2. Terminology . . . . . 4
2. High-Level State Option Extension . . . . . 4
2.1. High-level State Option Definition . . . . . 4
2.2. Using the High-level State Option . . . . . 5
2.2.1. Creating State Resources . . . . . 5
2.2.2. Querying State Resources . . . . . 9
2.2.3. Deleting States Resources . . . . . 10
3. Examples of Usage . . . . . 10
3.1. Example 1 . . . . . 10
3.2. Example 2 . . . . . 13
4. Security Considerations . . . . . 15
5. IANA Considerations . . . . . 16
6. Acknowledgements . . . . . 16
7. References . . . . . 16
7.1. Normative reference . . . . . 16
7.2. Informative Reference . . . . . 16
Appendix A. XML Schema for XML Serialization of One State Resources . . . . . 16
Appendix B. XML Schema for XML Serialization of Multiple State Resources . . . . . 18
Appendix C. Changelog . . . . . 19
Author's Address . . . . . 19

## 1. Introduction

This document adds a new option to the Constrained Application Protocol (CoAP): High-Level State.

### 1.1. Motivation

The Constrained Application Protocol [I-D.ietf-core-coap] (CoAP) is a lightweight efficient variant of the well-known Hypertext Transfer Protocol specifically designed for devices with limited resources such as small memory, little processing power, and constrained energy capacities. The main area of operation of CoAP is on wireless sensor nodes, i.e., wireless devices equipped with sensors to monitor environmental parameters such as temperature, air quality, or humidity. Hence, not only static metadata but also sensor values are retrieved by users via CoAP. The change frequency of measured sensor values depends on the one hand on the accuracy of the sensor but on the other hand on the (physical) property the sensor measures. It therefore may vary from milliseconds up to minutes, hours, or even days. A user interested in a parameter needs to request the current value periodically to keep track of changes. However, periodic querying can consume a good portion of the total amount of energy available and results in the quick depletion of a device's energy. Additionally, many requests might be unnecessary because the sensor value did not change compared to the last request. For that reason, CoAP observe [I-D.ietf-core-observe] introduces a mechanism to register interest in a resource much like with publish-subscribe systems. A CoAP server then only sends a response whenever the sensor value changes. As a result, only a reduced number of messages are required to keep track of the sensor readings.

Although the CoAP observe option can save resources, it might happen that the number of messages and thus, resource consumption even increases. This happens if the sensor value changes very often resulting in frequent update messages. Besides that, some clients may not be interested in raw precise sensor values but in a range a sensor values falls into. We call this range a high-level state because it categorizes the sensor value. So, instead of being interested if a room has 21.9 oC or 22.1 oC, the user might only want to know if it is "warm" in that room. The number of states for an environmental parameter is typically low. Accordingly, states change with much lower frequency as the underlying raw sensor values. Consequently, this leads to fewer responses when using CoAP observe.

The High-Level State option allows creating, querying and deleting high-level state resources for sensor resources on CoAP servers with the known CoAP request methods GET, POST, and DELETE. The user can define which sensor values are mapped to which high-level state.

Additionally, he can retrieve descriptions of already existing high-level state resources to reuse them.

### 1.2. Terminology

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

This document uses terms of the Constrained Application Protocol as defined in the terminology section of [I-D.ietf-core-coap].

Additionally, this specification defines the following terms:

High-level state:

> In contrast to a raw sensor reading a high-level state combines a number of sensor outputs under a new descriptive term given as a string.

High-level state resource:

> A CoAP resource, which returns different high-level states. State resource will be used synonymously with the term high-level state resource.

## 2. High-Level State Option Extension

### 2.1. High-level State Option Definition

| Type | C | U | N | R | Name | Format | Length | Default |
|---|---|---|---|---|---|---|---|---|
| TBD | - | - | - | x | High-Level State | (see below) | 1-257 B | (none) |

Figure 1: High-Level State Option Definition

The High-Level State Option is "elective" and "proxy-safe". It is "repeatable". Hence, the High-Level State Option can occur more than once. The use of repetition will be described in the following sections.

This Option can only be present in requests. Additionally, it has different semantics when used with different request methods. These

are described in the following sections.

The value carried in the Option has the following general format:

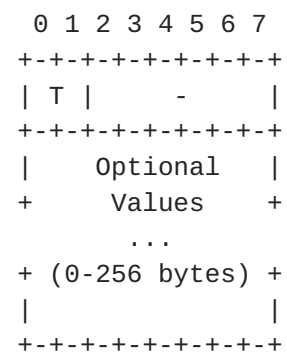


Figure 2: General format

T (TYPE): The value of the TYPE field is a 2 bit integer. If used in POST requests, it indicates the value format after the first byte. If used in GET requests, it specifies what representation a client expects in response to that request. Further details are discussed in Section 2.2.1 and Section 2.2.2 about the creation respectively querying of state resources.

The bits 2-7 are unused and MUST be ignored by the server.

## 2.2. Using the High-level State Option

The semantics of the Option depend on the used CoAP request method. In short, POST and DELETE create and remove state resources while GET is used to retrieve the state or a description of existing state resources.

### 2.2.1. Creating State Resources

To create a new state resource for a sensor resource, the client has to POST a request to the Uri-Path of the sensor. The server MUST create the state resource as a subresource of the sensor resource.

The output of most sensors is on a continuous numerical scale. However, some sensors output string or Boolean data types (true/ false respectively 1/0). The client should be able to map each of these data types to high-level states. For numerical values, the client should be able to specify mappings from intervals to states and for strings it should be able to map one or several different strings to a state. Boolean values can be easily mapped by using either the string mapping if the output is true or false or the numerical mapping if the output is 1 or 0. Hence, two formats for

the Option are available.

The used format is indicated by the TYPE field. Figure 3 shows the data types of a sensor output along with the integer used for the TYPE field and the format which is assumed to follow. Boolean is not listed because, as argued before, it can be mapped with the integer or string type.

| Data type | TYPE | Format |
|---|---|---|
| integer | 0 | 1 |
| float | 1 | 1 |
| string | 2 | 2 |

Figure 3: Format types

The number of bytes used for values in the different Option formats is given in Figure 4. Floats are encoded in Single-precision floating-point format as defined in [IEEE754]. One Option defines one state. Therefore, by repeating the High-Level State Option several states can be defined.

| Data type | Length (bytes) |
|---|---|
| integer | 2 |
| float | 4 |
| string | 0-128 |

Figure 4: Data type lengths

Format 1 is used for a mapping of an interval of numerical values to a state. It consists of a numerical lower and a numerical upper bound as well as a string giving the state name. The lower bound is inclusive while the upper bound is exclusive. This allows defining consecutive, continuous, non-intersecting intervals.

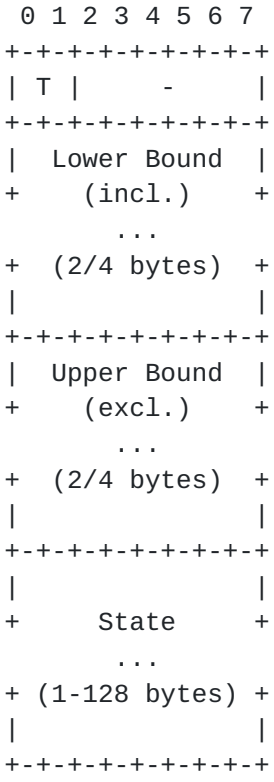


Figure 5: Format 1: Used for an interval mapping

Format 2 is to map a string to a state. Whenever the string output of the sensor matches the string which is given as the first parameter the string given as the second parameter is the current state.

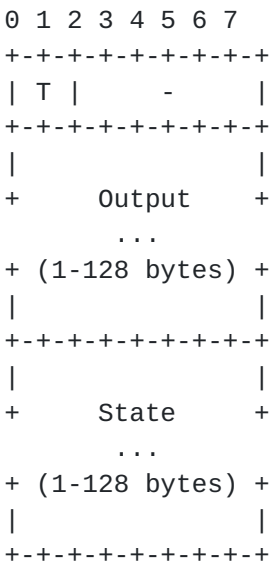


Figure 6: Format 2: Used for a string mapping

The server MUST ignore payload enclosed in the request.

If the server successfully created the state resource serving the defined states, it MUST send a response with response code 2.01 (Created) and the Location-Path Option which gives the relative Uri-Path for the newly created resource. The Uri-Path of the created resource can be an arbitraly allowed string. However, it is RECOMMENDED to use short Uri-Paths.

As already mentioned, by repeating the Option, several states can be defined. However, if the Option is repeated with different values for the TYPE field, the server MUST NOT process the request, and MUST send a response with response code 4.02 (Bad Option).

If the upper bound of an interval is smaller than the lower bound of that interval, the server MUST NOT process the request, and MUST send a response with response code 4.02 (Bad Option).

Due to ambiguity, it is forbidden to define different states for the same value. Hence, if a numerical mapping is used and if at least one intersection of any two intervals is non-empty, the server MUST NOT process the request, and MUST send a response with response code 4.02 (Bad Option). The same holds, if a string mapping is used and the same string is mapped to different states.

In the case that a client requests creation of a state resource with exactly same mappings and states as an already existing one, the server SHOULD send a response with response code 2.05 (Content) with the Uri-Path of the existing state resource as payload instead of creating another resource with the same semantics. Implementers must be aware that state resources can be deleted anytime. Accordingly, if a state resource is used by several clients and one deletes it, the other clients are not aware of that. Contrary, creating state resources with same semantics for each client, consumes more resources.

If the server is not able to create the state resource for the given sensor resource, e.g., because of insufficient resources, it MUST send a response with response code 5.03 (Service Unavailable). The server SHOULD include a payload indicating the reason for not creating the state resource.

If a client requests to create a state resource for a non-sensor resource, the server MUST NOT process the request and MUST send a response with response code 4.03 (Forbidden).

If the TYPE of a request is not matching the data type outputted by the sensor, the server SHOULD reject the request and SHOULD send a response with response code 4.02 (Bad Option).

### 2.2.2. Querying State Resources

The current state of a state resource can be retrieved by a normal GET-request without the High-Level State Option present in the request. However, by including the Option, the client can control the data which should be returned. The type of data that the server should return is indicated by the TYPE field in the request. Figure 7 gives an overview of all available T values.

| T | No. |
|---|---|
| State | 0 |
| State Number | 1 |
| Description | 2 |

Figure 7: Response types

If T = 0, the server MUST return the current state. With T = 1 an integer representing the state MUST be returned by the server. The integer is determined as follows: in each state resource creation

process upon a POST-request the states MUST be enumerated starting with 0 by their order of appearance in the High-Level State Options. If T = 2 the server MUST return a description of the state resource describing the mappings and the appropriate states. An XML Schema for the XML format is given in Appendix A. If there is no defined state for a sensor value, i.e., there is no mapping for this value, the server MUST return the state "undefined" if T = 0 and -1 if T = 1.

A list of all available state resource of a sensor resources can be retrieved by sending a GET-request with High-Level-State Option to the sensor resource with T = 2 included. The xml format the server MUST return is defined by the XML Schema in Appendix B. All other requests with the High-Level State Option to a sensor resource SHOULD be ignored by the server.

### 2.2.3. Deleting States Resources

If a client wants to remove a state resource, it has to send a DELETE message to the state resource. The server MUST delete the state resource and a "2.02 (Deleted) response code SHOULD be used on success or in case the resource did not exist before the request" as stated in Section 5.8.4 in [I-D.ietf-core-coap].

## 3. Examples of Usage

In the following sections two examples show how the high-level state option is used to create, query, and delete state resources.

### 3.1. Example 1

Consider four users who want to retrieve data from a temperature sensor. However, they are not interested in the raw values but high-level states. Hence, they want to create state resources on the server. The parameters of the High-Level State Option in the examples are T and the appropriate optional values as described in Section 2.2.

User 1 is the first to communicate with the CoAP server. He wants to have two states, namely a "cold" and a "warm" state, where the first is defined as temperature between -50 oC and 20 oC and the second all values which are between 20 oC and 50 oC. For that, a POST-request with the bounds is send to the server:

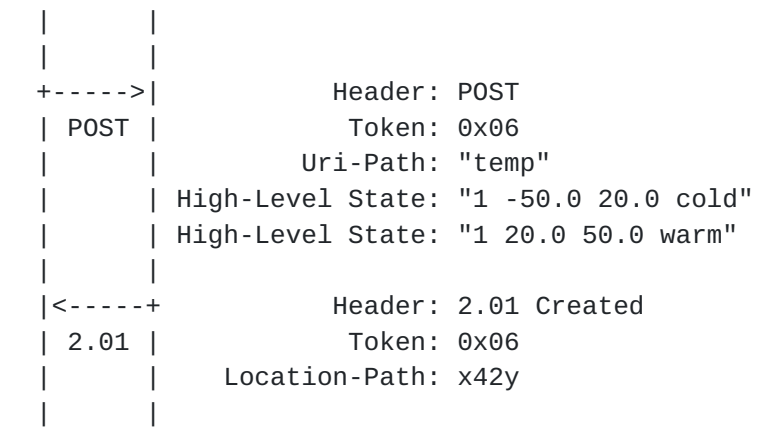


Figure 8: User 1 creating a state resource with two states

The server answers with a response code of 2.01 (Created) and a Location-Path indicating that a state resource, which can serve the two desired high-level states, is now available under the given path.

Second, the next user wants to have four states ("cold", "moderate", "warm", and "hot") with bounds of -50 oC, 0 oC, 10 oC, 25 oC, and 50 oC. The communication to create these states looks as follows:

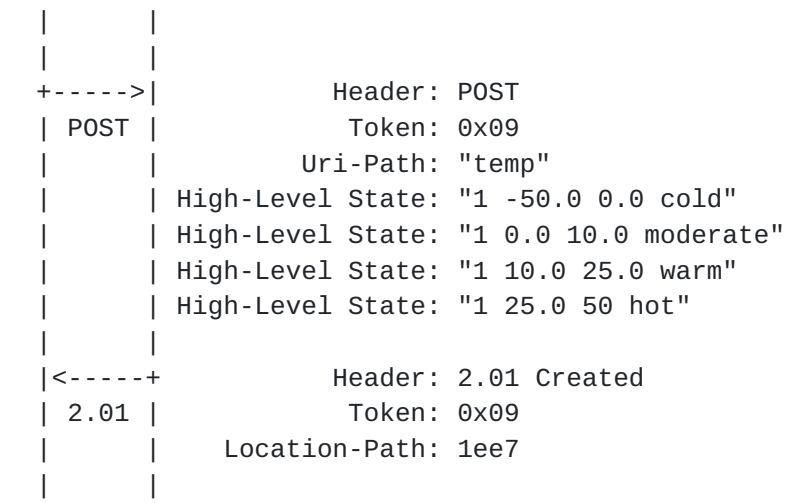


Figure 9: User 2 creating another state resource

Afterwards, the server has two state resources for the sensor resource. The one serves two states and the other one which serves four states.

Also the third user wants to create a high-level resource. By chance, he requests the same states and bounds as the first user which can be seen in the POST-request:

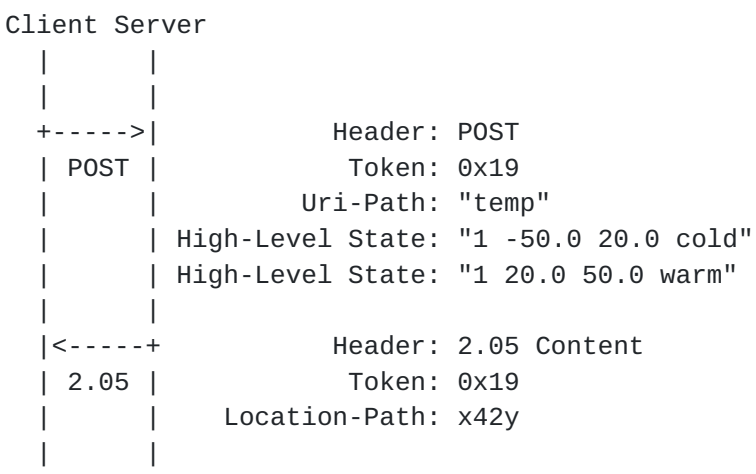


Figure 10: Creation of a State Resource with Same Semantics as Already Existing State Resource

Consequently, the response is different too the one sent to user 1. The response code is 2.04 (Changed) and the Location-Path Option gives the path of the already existing resource.

The last user finally sends his POST-request to create another resource with three states:

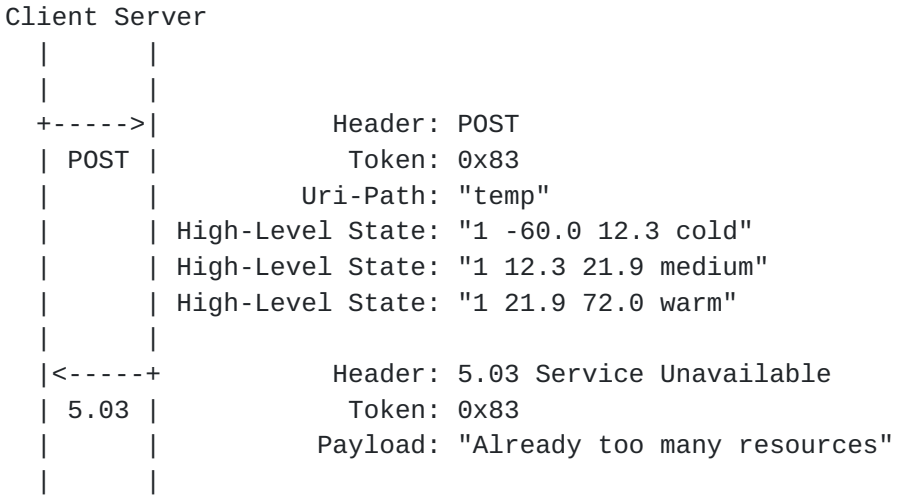


Figure 11: Failing of State Resource Creation due to Low Resources

Unfortunately, the server rejected to create a new resource due to insufficient resources which is indicated by the response. Because of that the user sent another request to retrieve a list of already available state resources:

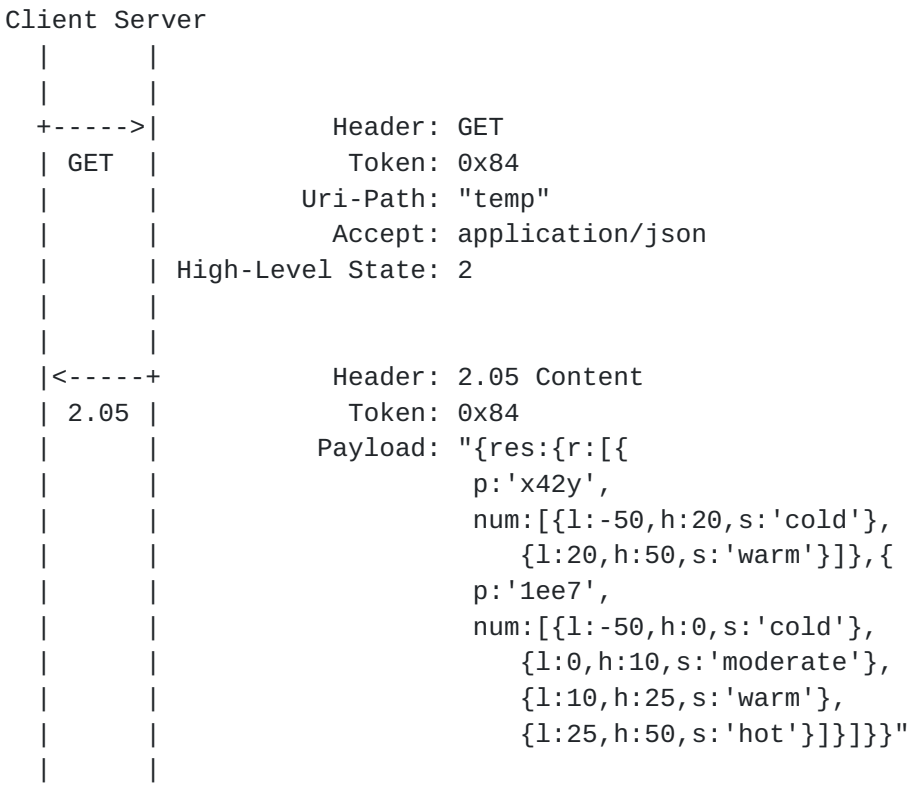


Figure 12: Retrieving List of Existing State Resources

With this list of JSON-encoded state resource the user has the ability to decide if he wants to use one of the existing state resources.

## 3.2. Example 2

In the second example, we consider a sensor which outputs strings describing the current weather and a user who first wants to use a string mapping to create a state resource for the weather sensor. Afterwards, he queries the state resource and finally deletes it. The output range of the sensor is "rainy", "cloudy", "sunny", and "foggy".

The client sends a POST-request with a string mapping to create the two states "home" and "beach".

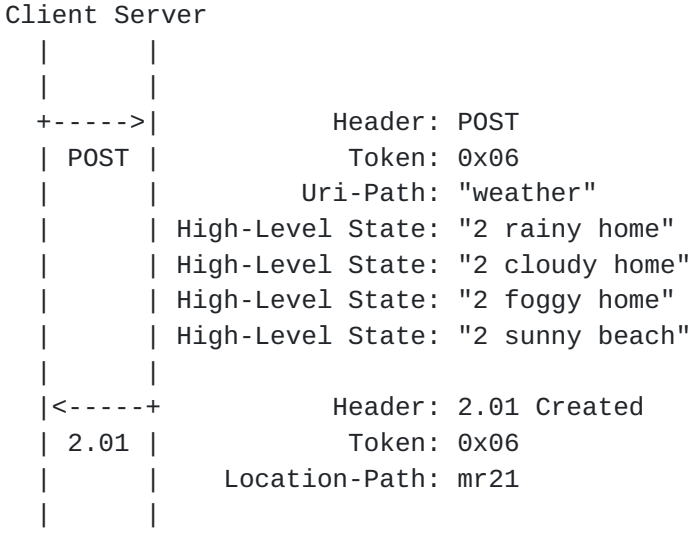


Figure 13: The User Creates a State Resource

After creation of the state resource, he retrieves the current state by sending a GET-request.

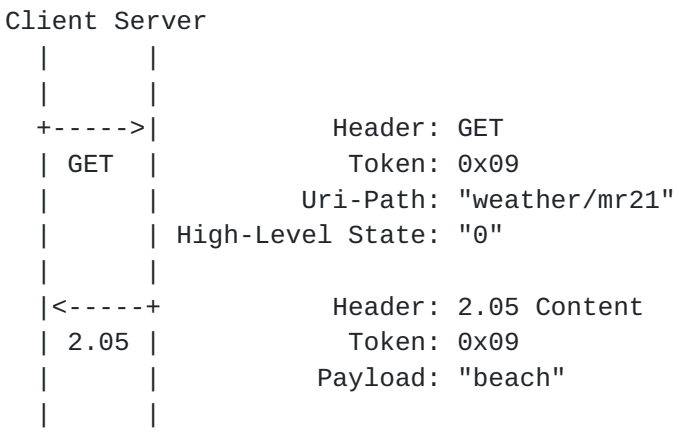


Figure 14: The User Retrieves the Current State

Because it is good weather, the user decides to go to the beach. But before, he releases the resources on the server by deleting the state resource.

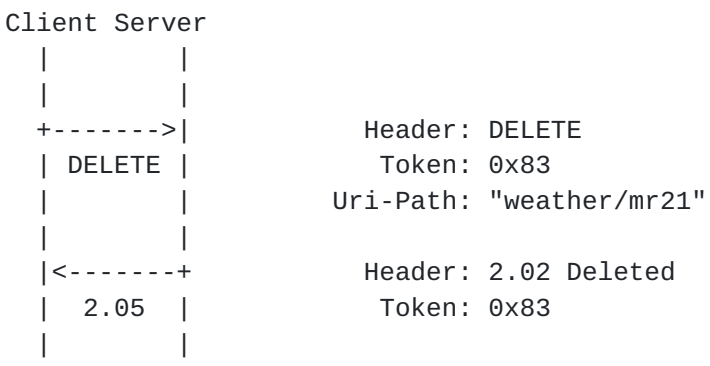


Figure 15: The User Deletes the State Resource

## 4. Security Considerations

PUT operations, often used for updates, in conjunction with the High-Level State Option are forbidden. Hence, the server MUST NOT process such requests and MUST respond with a response code of 4.05 (Method not allowed). Updates of state resources can lead to unexpected behavior of clients if several clients use the same state resource. If one client is updating mappings or states of a state resource, other clients which are not aware of the update could end up in abnormal behavior because they cannot handle the unexpected results. For the same reason, a server SHOULD NOT reuse Uri-Paths of deleted state resources.

Depending on the implementation and the remaining resources of the server, creation of state resources can consume a considerable amount of resources. However, this is true for all resource creations and not limited to the presented new High-Level State Option. Anyway, implementers SHOULD be aware of this fact and consider countermeasures such as limiting the number of state resources which can be created, limiting the number of state resource creation requests per client, or to introduce a duration after a successful state resource creation in which further requests are rejected.

## 5. IANA Considerations

The IANA is requested to add the following "CoAP Option Numbers" entry as per Section 12.2 of [I-D.ietf-core-coap]:

| Number | Name | Reference |
|---|---|---|
| TBD | High-Level State | Section 2 of this document |

## 6. Acknowledgements

Thanks to Lukas Ruge, Dennis Pfisterer, Kay Roemer and Philipp Abraham for proof-reading, helpful comments and discussions that have helped to shape this document.

## 7. References

### 7.1. Normative reference

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

[IEEE754] Institute of Electrical and Electronics Engineers (IEEE), "754-2008 - IEEE Standard for Floating-Point Arithmetic", August 2008.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

### 7.2. Informative Reference

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-11 (work in progress), October 2013.

## Appendix A. XML Schema for XML Serialization of One State Resources

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.example.com/state-option"
           targetNamespace="http://www.example.com/state-option"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">

 <!-- To describe a state resources -->
 <xs:element name="r" type="resource"/>

 <!-- To describe the mappings of a state resource along
      with the uri-path-->
 <xs:complexType name="resource">
  <xs:sequence>
   <xs:choice>
    <xs:element name="num" type="num_map" maxOccurs="unbounded"/>
    <xs:element name="str" type="string_map" maxOccurs="unbounded"/>
   </xs:choice>
  </xs:sequence>
 </xs:complexType>

<!-- To describe a numerical mapping with lower bound,
     upper bound and state -->
 <xs:complexType name="num_map">
  <xs:sequence>
   <xs:sequence maxOccurs="unbounded">
    <xs:element name="l" type="xs:float"/>
    <xs:element name="h" type="xs:float"/>
   </xs:sequence>
   <xs:element name="s" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>

 <!-- To describe a string mapping with
      one or more strings and state -->
 <xs:complexType name="string_map">
  <xs:sequence>
   <xs:element name="str" type="xs:string" maxOccurs="unbounded"/>
   <xs:element name="s" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>

</xs:schema>
```

## Appendix B. XML Schema for XML Serialization of Multiple State Resources

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns="http://www.example.com/state-option"
           targetNamespace="http://www.example.com/state-option"
           elementFormDefault="qualified"
           attributeFormDefault="unqualified">

 <xs:element name="res" type="res"/>

 <!-- To describe one or more state resources -->
 <xs:complexType name="res">
  <xs:sequence>
   <xs:element name="r" type="resource" maxOccurs="unbounded"/>
  </xs:sequence>
 </xs:complexType>

 <!-- To describe the mappings of a state resource along
      with the uri-path-->
 <xs:complexType name="resource">
  <xs:sequence>
   <xs:element name="p" type="xs:normalizedString"/>
   <xs:choice>
    <xs:element name="num" type="num_map" maxOccurs="unbounded"/>
    <xs:element name="str" type="string_map" maxOccurs="unbounded"/>
   </xs:choice>
  </xs:sequence>
 </xs:complexType>

 <!-- To describe a numerical mapping with lower bound,
      upper bound and state -->
 <xs:complexType name="num_map">
  <xs:sequence>
   <xs:sequence maxOccurs="unbounded">
    <xs:element name="l" type="xs:float"/>
    <xs:element name="h" type="xs:float"/>
   </xs:sequence>
   <xs:element name="s" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>

 <!-- To describe a string mapping with
      one or more strings and state -->
 <xs:complexType name="string_map">
  <xs:sequence>
```

```
      <xs:element name="str" type="xs:string" maxOccurs="unbounded"/>
      <xs:element name="s" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>

   </xs:schema>
```

## Appendix C. Changelog

Changes from draft-00 to draft-01:

- Changed contact details of author.
- Updated references of CoAP and CoAP observe.
- Fixed number of Bytes in Figure Figure 2 from 257 to 256.
- Fixed number of Bytes in Figure Figure 5 from 2-4 to 2/4.
- Fixed number of Bytes in Figures Figure 5 and Figure 6 from 0-128 to 1-128.
- Fixed tokens in Figure Figure 15 and Figure 14.
- Corrected caption of Figure Figure 15. Before it was the same as in Figure Figure 14 because of copy \& paste.
- Removed "or equal" from the paragraph where handling of a upper bound smaller than the lower bound is described (Section Section 2.2.1). Equal upper and lower bound are allowed because one is inclusive and the other exclusive.

## Author's Address

Richard Mietz
University of Luebeck
Ratzeburger Allee 160
Luebeck, Schleswig-Holstein 23562
DE

Phone: +49 451 500 5984
Email: mietz@itm.uni-luebeck.de
URI: http://www.itm.uni-luebeck.de/