Workgroup: CoRE Working Group
Internet-Draft:
draft-fossati-core-coap-problem-01
Published: 3 March 2020
Intended Status: Standards Track
Expires: 4 September 2020
Authors: T. Fossati
ARM
J. Jiménez
Ericsson

Problem Details For CoAP APIs

Abstract

This document defines a "problem detail" as a way to carry machine-readable details of errors in a CoAP response to avoid the need to define new error response formats for CoAP APIs.

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 4 September 2020.

Copyright Notice

Table of Contents

1. Introduction

CoAP [RFC7252] response codes are sometimes not sufficient to convey enough information about an error to be helpful.

This specification defines a simple and extensible CBOR [RFC7049] format to suit this purpose. It is designed to be reused by CoAP APIs, which can identify distinct "problem types" specific to their needs.

Thus, API clients can be informed of both the high-level error class (using the response code) and the finer-grained details of the problem (using this format).

The format presented is largely inspired by the Problem Details for HTTP APIs defined in [RFC7807].

1.1. Requirements Language

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

2. CoAP Problem Details Definition

A CoAP Problem Details is encoded as a CBOR map with the following members:

* "ns" (int) - A code-point that defines the namespace under which the "type" field needs to be interpreted. This is a mandatory field.

* "type" (uint) - A code-point that identifies the problem type within the namespace. This is a mandatory field.

* "title" (text) - A short, human-readable summary of the problem type. It SHOULD NOT change from occurrence to occurrence of the problem.

* "response-code" (8-bit uint) - The CoAP response code ([RFC7252], Section 5.9) generated by the origin server for this occurrence of the problem.

* "detail" (text) - A human-readable explanation specific to this occurrence of the problem.

* "instance" (uri) - A URI reference that identifies the specific occurrence of the problem. It may or may not yield further information if dereferenced.

Consumers MUST use "ns" and "type" as primary identifiers for the problem type; the "title" string is advisory and included only for consumers who are not aware of the semantics of the "ns" and "type" values.

The "response-code" member, if present, is only advisory; it conveys the CoAP response code used for the convenience of the consumer. Generators MUST use the same response code in the actual CoAP response, to assure that generic CoAP software that does not understand this format still behaves correctly. Consumers can use the response-code member to determine what the original response code used by the generator was, in cases where it has been changed (e.g., by an intermediary or cache), and when message payloads persist without CoAP information (e.g., in an events log or analytics database). Generic CoAP software will still use the CoAP response code.

The "detail" member, if present, ought to focus on helping the client correct the problem, rather than giving debugging information. Consumers SHOULD NOT parse the "detail" member for information; extensions (see Section 3.2) are more suitable and less error-prone ways to obtain such information.

Note that "instance" accepts relative URIs; this means that it must be resolved relative to the document's base URI, as per [RFC3986], Section 5.

2.1. CDDL

The definition in CDDL format [RFC8610] of a Problem Details for CoAP is provided in Figure 1.

```
coap-problem-details = {
  ns => int,
  type => uint,
  ? title => text,
  ? response-code => uint .size 1,
  ? detail => text,
  ? instance => uri,
  * $$coap-problem-details-extension,
}

ns = 0
type = 1
title = 2
response-code = 3
detail = 4
instance = 5
```

Figure 1: CoAP Problem Details: CDDL Definition

3. Extensibility

The format presented can be extended at two separate points that allow the definition of:

- New problem type values (see Section 3.1); and
- New problem attributes (see Section 3.2).

3.1. Defining New Problem Types

The mechanism for defining new problem types is designed to allow private use, for example by organisations or projects, while at the same time supporting the use of this error format in public protocols and APIs, as well as ease of transition between the two - for example if an API is first developed internally to an organisation and then open-sourced. Another critical design objective is to enable delegating the administration of the code-points space to entities (and experts) that are "closer" to the actual usage and intended meaning of the code-points. In fact, an explicit desiderata is to avoid having experts looking over a very big and diverse semantic space.

To meet these goal, new problem types are always defined (and have a meaning) within a namespace. The namespace range is itself partitioned in three separate sub-ranges: a completely private space, one devoted to private organisations and projects, and a third one used for public APIs and protocols. The rules for registering a new namespace are outlined in Section 5.2.2.

The registration procedures for new problem types are not defined in this document. At a minimum, though, new problem type definitions SHOULD document:

1. A parent namespace;

2. Their own code-point;

3. A title that appropriately describes the problem type (think short); and

4. The CoAP response-code for it to be used with.

A problem type definition may specify additional attributes on the problem details map (see Section 3.2).

(Note on renumbering: moving a set of error types from the private to the public space needs only changing the namespace identifier while leaving all error types the same.)

3.2. Defining New Problem Attributes

Problem type definitions MAY extend the problem details object with additional attributes to convey additional, problem-specific information.

Clients consuming problem details MUST ignore any such extensions that they do not recognize; this allows problem types to evolve and include additional information in the future.

CoAP Problem Details can be extended via the coap-problem-details-extension CDDL socket (see Section 3.9 of [RFC8610]).

4. Security Considerations

The security and privacy considerations outlined in Section 5 of [RFC7807] apply in full.

5. IANA Considerations

5.1. Registration of a Content-Format identifier for application/coap-problem+cbor

This document requests that IANA registers the following Content-Format to the "CoAP Content-Formats" sub-registry, within the "Constrained RESTful Environments (CoRE) Parameters" registry, from the Expert Review space (0..255):

Media Type	Encoding	ID	Reference
application/coap-problem+cbor	--	TBD1	RFCthis

Table 1

5.2. New Registries

This document requests that IANA create the following new registries:

*CoAP Problem Namespaces (Section 5.2.2);

*CoAP Problem Details (Section 5.2.1).

5.2.1. CoAP Problem Details Registry

The "CoAP Problem Details" registry keeps track of the allocation of the integer values used as index values in the coap-problem-details CBOR map.

Future registrations for this registry are to be made based on [RFC8126] as described in Table 2.

Range	Registration Procedures
0...N	Standards Action
N+1...4294967295	Specification Required

Table 2: CoAP Problem Details Registration Procedures

All negative values are reserved for Private Use.

Initial registrations for the "CoAP Problem Details" registry are provided in Table 3. Assignments consist of an integer index value, the item name, and a reference to the defining specification.

Index	Index Name	Specification
0	ns	RFCthis
1	type	RFCthis
2	title	RFCthis
3	response-code	RFCthis
4	detail	RFCthis
5	instance	RFCthis

Table 3: CoAP Problem Details Initial Registrations

5.2.2. CoAP Problem Namespace Registry

The "CoAP Problem Namespace" registry keeps track of the problem namespace values.

Future registrations for this registry are to be made based on [RFC8126] as described in Table 4.

Range	Registration Procedures
-L...-1	First Come First Served
0...M	Standards Action
M+1...4294967295	Specification Required

Table 4: CoAP Problem Types Registration Procedures

All negative values less than L are reserved for Private Use.

The "CoAP Problem Namespace" registry has three columns as shown in Table 5. Assignments consist of an integer index value, the item description, and a reference to the defining specification.

Value	Description	Specification
empty	empty	empty

Table 5: CoAP Problem Namespace Registry

The registry is initially empty.

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, <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>.

[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>.

[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>.

[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>.

6.2. Informative References

[I-D.ietf-core-coral] Hartke, K., "The Constrained RESTful Application Language (CoRAL)", Work in Progress, Internet-Draft, draft-ietf-core-coral-02, 8 January 2020, <http://www.ietf.org/internet-drafts/draft-ietf-core-coral-02.txt>.

[I-D.ietf-core-href] Hartke, K., "Constrained Resource Identifiers", Work in Progress, Internet-Draft, draft-ietf-core-href-02, 8 January 2020, <http://www.ietf.org/internet-drafts/draft-ietf-core-href-02.txt>.

[RFC7807] Nottingham, M. and E. Wilde, "Problem Details for HTTP APIs", RFC 7807, DOI 10.17487/RFC7807, March 2016, <https://www.rfc-editor.org/info/rfc7807>.

Appendix A. Examples

This section presents a series of examples in CBOR diagnostic notation [RFC7049]. The examples are fictitious. No identification with actual products is intended or should be inferred. All examples involve the same CoAP problem type (5, with pretend semantics "unknown key id") defined in the private namespace "-33455".

A.1. Minimalist

The example in Figure 2 has the most compact representation. By avoiding any non-mandatory field, the Problem encodes to seven bytes in total. This is suitable for a constrained receiver that happens to have precise knowledge of the semantics associated with the namespace and type code-points.

```
{
    / ns /   0: -33455, / a private namespace /
    / type / 1: 5       / "unknown key id" semantics /
}
```

Figure 2: Private Namespace: Minimal Payload

A.2. Full-Fledged

The example in Figure 3 has all the mandatory as well as the optional fields populated. This format is appropriate for an unconstrained receiver. For example, an edge gateway forwarding to a log server that needs to gather as much contextual information as possible, including details about the error condition, the associated CoAP response code, and even the URL describing the specific error instance.

```
{
    / ns /            0: -33455,
    / type /          1: 5,
    / title /         2: "unknown key id",
    / response-code / 3: 132, / 4.04 Not Found /
    / detail /        4: "Key with id 0x01020304 not registered",
    / instance /      5: 32("https://private-api.example/errors/5")
}
```

Figure 3: Private Namespace: Full Payload

A.3. Full-Fledged with Extensions

The last example (Figure 4) makes use of the built-in extension mechanism described in Section 3.2 to provide some context specific information - in this made up scenario a list of possible key ids is provided to the receiving end. This richer format might be enabled for debug or tracing purposes, possibly on a per-transaction basis. Note that the map index for key-ids key is minted from the private (negative) space.

```
{
    / ns /            0: -33455,
    / type /          1: 5,
    / title /         2: "unknown key id",
    / response-code / 3: 132, / 4.04 Not Found /
    / detail /        4: "Key with id 0x01020304 not registered",
    / instance /      5: 32("https://private-api.example/errors/5"),
    / key-ids /       -1: [ 0x01020300, 0x01020301, 0x01020302 ]
}
```

Figure 4: Private Namespace: Full Payload and Extension

Appendix B. Doing it with CoRAL

CoRAL [I-D.ietf-core-coral] provides a way to address the same problem that is solved by the format described in this document. (Refer to section 5.2.3 of [I-D.ietf-core-coral] for initial discussion around CoRAL Error Responses.)

By abstracting the serialization aspects (CBOR, JSON, etc.), the transport protocol (HTTP, CoAP, etc.) and its response codes, while also providing compression of the involved resources, CoRAL can potentially support a more general solution than the one discussed here, in particular one that also supersedes [RFC7807].

B.1. Examples

In this section, the examples from Appendix A are converted to CoRAL.

The main differences are:

* CoRAL is using an array of alternating keys and values instead of a map with array values to get a multi-dict;

* CoRAL uses [I-D.ietf-core-href] as an alternative to URIs that is optimized for constrained nodes;

* CoRAL uses its own code-point allocation scheme.

B.1.1. Minimalist

* Textual format:

```
#using <http://example.org/vocabulary/problem-details#>
#using ex = <http://vocabulary.private-api.example/#>

type            ex:unknown-key-id
```

* CBOR serialisation:

```
[
    / type / 1, 5       / "unknown key id" semantics /
]
```

B.1.2. Full-Fledged

* Textual format:

```
#using <http://example.org/vocabulary/problem-details#>
#using ex = <http://vocabulary.private-api.example/#>

type            ex:unknown-key-id
title           "unknown key id"
response-code   132
detail          "Key with id 0x01020304 not registered"
instance        <https://private-api.example/errors/5>
```

*CBOR serialisation:

```
[
    / type /          1, 5,
    / title /         2, "unknown key id",
    / response-code / 3, 132, / 4.04 Not Found /
    / detail /        4, "Key with id 0x01020304 not registered",
    / instance /      5, [1, "https",
                          2, "private-api.example",
                          6, "errors",
                          6, "5"]
]
```

B.1.3. Full-Fledged with Extensions

*Textual format:

```
#using <http://example.org/vocabulary/problem-details#>
#using ex = <http://vocabulary.private-api.example/#>

type            5
title           "unknown key id"
response-code   132
detail          "Key with id 0x01020304 not registered"
instance        <https://private-api.example/errors/5>
ex:key-id       0x01020300
ex:key-id       0x01020301
ex:key-id       0x01020302
```

*CBOR serialisation:

```
[
    / type /          1, 5,
    / title /         2, "unknown key id",
    / response-code / 3, 132, / 4.04 Not Found /
    / detail /        4, "Key with id 0x01020304 not registered",
    / instance /      5, [1, "https",
                          2, "private-api.example",
                          6, "errors",
                          6, "5"],
    / key-id /      100, 0x01020300,
    / key-id /      100, 0x01020301,
    / key-id /      100, 0x01020302
]
```

Appendix C. Contributors

Klaus Hartke provided the text in Appendix B.1.

Acknowledgments

Mark Nottingham and Erik Wilde, authors of RFC 7807. Carsten Bormann and Klaus Hartke for discussion on the problem space and extensibility requirements.

Authors' Addresses

Thomas Fossati
ARM

Email: thomas.fossati@arm.com

Jaime Jiménez
Ericsson

Email: jaime@iki.fi