Workgroup: CoRE Working Group
Internet-Draft:
draft-ietf-core-problem-details-07
Published: 28 June 2022
Intended Status: Standards Track
Expires: 30 December 2022
Authors: T. Fossati
arm
C. Bormann
Universität Bremen TZI

Concise Problem Details For CoAP APIs

Abstract

This document defines a concise "problem detail" as a way to carry machine-readable details of errors in a REST response to avoid the need to define new error response formats for REST APIs for constrained environments. The format is inspired by, but intended to be more concise than, the Problem Details for HTTP APIs defined in RFC 7807.

About This Document

This note is to be removed before publishing as an RFC.

Status information for this document may be found at https://datatracker.ietf.org/doc/draft-ietf-core-problem-details/.

Discussion of this document takes place on the Constrained RESTful Environments Working Group mailing list (mailto:core@ietf.org), which is archived at https://mailarchive.ietf.org/arch/browse/core/.

Source for this draft and an issue tracker can be found at https://github.com/core-wg/core-problem-details.

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 30 December 2022.

Table of Contents

1. Introduction

REST response status information such as CoAP response codes (Section 5.9 of [RFC7252]) is sometimes not sufficient to convey enough information about an error to be helpful. This specification

defines a simple and extensible framework to define CBOR [STD94] data items to suit this purpose. It is designed to be reused by REST APIs, which can identify distinct "shapes" of these data items 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 the vocabulary defined here). This pattern of communication is illustrated in Figure 1.

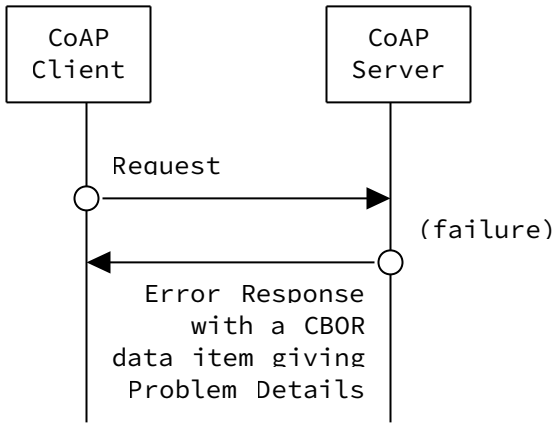

Figure 1: Problem Details: Example with CoAP

The framework presented is largely inspired by the Problem Details for HTTP APIs defined in [RFC7807]. Appendix B discusses applications where interworking with [RFC7807] is required.

1.1. Terminology and Requirements Language

The terminology from [RFC7252], [STD94], and [RFC8610] applies; in particular CBOR diagnostic notation is defined in Section 8 of [STD94] and Appendix G of [RFC8610]. Readers are also expected to be familiar with the terminology from [RFC7807].

In this document, the structure of data is specified in CDDL [RFC8610] [RFC9165].

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. Basic Problem Details

A Concise Problem Details data item is a CBOR data item with the following structure (rules named starting with tag38 are defined in Appendix A):

```
problem-details = non-empty<{
  ? &(title: -1) => oltext
  ? &(detail: -2) => oltext
  ? &(instance: -3) => ~uri
  ? &(response-code: -4) => uint .size 1
  ? &(base-uri: -5) => ~uri
  ? &(base-lang: -6) => tag38-ltag
  ? &(base-rtl: -7) => tag38-direction
  standard-problem-detail-entries
  custom-problem-detail-entries
}>

standard-problem-detail-entries = (
  * nint => any
)

custom-problem-detail-entries = (
  * (uint/~uri) => { + any => any }
)

non-empty<M> = (M) .and ({ + any => any })

oltext = text / tag38
```

Figure 2: Structure of Concise Problem Details Data Item

(Examples of elaborated Concise Problem Details data items can be found later in the document, e.g., Figure 3.)

A number of problem detail entries, the Standard Problem Detail entries, are predefined (more predefined details can be registered, see Section 3.1).

Note that, unlike [RFC7807], Concise Problem Details data items have no explicit "problem type". Instead, the category (or, one could say, Gestalt) of the problem can be understood from the shape of the problem details offered. We talk of a "problem shape" for short.

The title (key -1):
: A short, human-readable summary of the problem shape. Beyond the shape of the problem, it is not intended to summarize all the specific information given with the problem details. For

instance, the summary might include that an account does not have enough money for a transaction to succeed, but not the detail information such as the account number, how much money that account has, and how much would be needed.

The detail (key -2):
A human-readable explanation specific to this occurrence of the problem.

The instance (key -3):
A URI reference that identifies the specific occurrence of the problem. It may or may not yield further information if dereferenced.

The response-code (key -4):
The CoAP response code (Sections 5.9 and 12.1.2 of [RFC7252]) generated by the origin server for this occurrence of the problem.

The base-uri (key -5):
The base URI (Section 5.1 of [STD66]) that should be used to resolve relative URI references embedded in this Concise Problem Details data item.

The base-lang (key -6):
The language-tag (tag38-ltag) that applies to the presentation of unadorned text strings (not using tag 38) in this Concise Problem Details data item, see Appendix A.

The base-rtl (key -7):
The writing-direction (tag38-direction) that applies to the presentation of unadorned text strings (not using tag 38) in this Concise Problem Details data item, see Appendix A.

Both "title" and "detail" can use either an unadorned CBOR text string (text) or a language-tagged text string (tag38); see Appendix A for the definition of the latter. Language tag and writing direction information for unadorned text strings are intended to be obtained from context; if that context needs to be saved or forwarded with a Concise Problem Details data item, "base-lang" and "base-rtl" can be used for that. If no such (explicitly saved or implicit) context information is available, unadorned text is interpreted with language-tag "en" and writing-direction "false" (ltr).

The "title" string is advisory and included to give consumers a shorthand for the category (problem shape) of the error encountered.

The "detail" member, if present, ought to focus on helping the client correct the problem, rather than giving extensive server-side

debugging information. Consumers **SHOULD NOT** parse the "detail" member for information; extensions (see Section 3) are more suitable and less error-prone ways to obtain such information. Note that the "instance" URI reference may be relative; this means that it must be resolved relative to the representation's base URI, as per Section 5 of [STD66].

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 here as in the actual CoAP response; the latter is needed to assure that generic CoAP software that does not understand the problem-details 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 bodies persist without CoAP information (e.g., in an events log or analytics database). Generic CoAP software will still use the CoAP response code. To support the use case of message body persistence without support by the problem-details generator, the entity that persists the Concise Problem Details data item can copy over the CoAP response code that it received on the CoAP level. Note that the "response-code" value is a numeric representation of the actual code (see Section 3 of [RFC7252]), so it does not take the usual presentation form that resembles an HTTP status code -- 4.04 Not found is represented by the number 132.

The "base-uri" member is usually not present in the initial request-response communication as it can be inferred as per Section 5.1.3 of [STD66]. An entity that stores a Concise Problem Details data item or otherwise makes it available for consumers without this context might add in a base-uri member to allow those consumers to perform resolution of any relative URI references embedded in the data item.

3. Extending Concise Problem Details

This specification defines a generic problem details container with only a minimal set of attributes to make it usable.

It is expected that applications will extend the base format by defining new attributes.

These new attributes fall into two categories: generic and application specific.

Generic attributes will be allocated in the standard-problem-detail-entries slot according to the registration procedure defined in Section 3.1.

Application-specific attributes will be allocated in the custom-problem-detail-entries slot according to the procedure described in Section 3.2.

Consumers of a Concise Problem Details data item **MUST** ignore any Standard or Custom Problem Detail entries, or keys inside the Custom Problem Detail entries, that they do not recognize ("ignore-unknown rule"); this allows problem details to evolve. When storing the data item for future use or forwarding it to other consumers, it is strongly **RECOMMENDED** to retain the unrecognized entries; exceptions might be when storage/forwarding occurs in a different format/protocol that cannot accommodate them, or when the storage/forwarding function needs to filter out privacy-sensitive information and for that needs to assume unrecognized entries might be privacy-sensitive.

3.1. Standard Problem Detail Entries

Beyond the Standard Problem Detail keys defined in Figure 2, additional Standard Problem Detail keys can be registered for use in the standard-problem-detail-entries slot (see Section 6.1).

Standard Problem Detail keys are negative integers, so they can never conflict with Custom Problem Detail keys defined for a specific application domain (which are unsigned integers or URIs.)

In summary, the keys for Standard Problem Detail entries are in a global namespace that is not specific to a particular application domain.

3.1.1. Standard Problem Detail Entry: Unprocessed CoAP Option

Section 2 provides a number of generally applicable Standard Problem Detail Entries. The present section both registers another useful Standard Problem Detail entry and serves as an example of a Standard Problem Detail Entry registration, in the registration template format that would be ready for registration.

> **Key Value:** TBD (assigned at registration)
>
> **Name:** unprocessed-coap-option
>
> **CDDL type:** one-or-more<uint>, where
>
> ```
> one-or-more<T> = T / [ 2* T ]
> ```
>
> **Brief description:** Option number(s) of CoAP option(s) that were not understood
>
> **Specification reference:** Section 3.1.1 of RFC XXXX

RFC Editor: please replace RFC XXXX with the RFC number of this RFC and remove this note.

The specification of the Standard Problem Detail entry referenced by the above registration template follows:

The Standard Problem Detail entry unprocessed-coap-option provides the option number(s) of CoAP option(s) present in the request that could not be processed by the server.

This may be a critical option that the server is unaware of, or an option the server is aware of but could not process (and chose not to, or was not allowed to, ignore it).

The Concise Problem Details data item including this Standard Problem Detail Entry can be used in fulfillment of the "**SHOULD**" requirement in Section 5.4.1 of [RFC7252].

Several option numbers may be given in a list (in no particular order), without any guarantee that the list is a complete representation of all the problems in the request (as the server might have stopped processing already at one of the problematic options). If an option with the given number was repeated, there is no indication which of the values caused the error.

Clients need to expect seeing options in the list they did not send in the request; this can happen if the request traversed a proxy that added the option but did not act on the problem details response being returned by the origin server.

Note that for a few special values of unprocessed CoAP options (such as Accept or Proxy-Uri), there are special response codes (4.06 Not Acceptable, 5.05 Proxying Not Supported, respectively) to be sent instead of 4.02 Bad Option.

3.2. Custom Problem Detail Entries

Applications may extend the Problem Details data item with additional entries to convey additional, application-specific information.

Such new entries are allocated in the custom-problem-detail-entries slot, and carry a nested map specific to that application. The map key can either be an (absolute!) URI (under control of the entity defining this extension), or an unsigned integer. Only the latter needs to be registered (Section 6.2).

Within the nested map, any number of attributes can be given for a single extension. The semantics of each custom attribute **MUST** be described in the documentation for the extension; for extensions

that are registered (i.e., are identified by an unsigned int) that documentation goes along with the registration.

The unsigned integer form allows a more compact representation. In exchange, authors are expected to comply with the required registration and documentation process. In comparison, the URI form is less space-efficient but requires no registration. It is therefore useful for experimenting during the development cycle and for applications deployed in environments where producers and consumers of Concise Problem Details are more tightly integrated. (The URI form thus covers the potential need we might otherwise have for a "private use" range for the unsigned integers.)

Note that the URI given for the extension is for identification purposes only and, even if dereferenceable in principle, it **MUST NOT** be dereferenced in the normal course of handling problem details (i.e., outside diagnostic/debugging procedures involving humans).

Figure 3 shows an example (in CBOR diagnostic notation) of a custom extension using a (made-up) URI as custom-problem-detail-entries key.

```
{
  / title /         -1: "title of the error",
  / detail /        -2: "detailed information about the error",
  / instance /      -3: "coaps://pd.example/FA317434",
  / response-code / -4: 128, / 4.00 /

  "tag:3gpp.org,2022-03:TS29112": {
    / cause /  0: "machine-readable error cause",
    / invalidParams / 1: [
      [
        / param / "first parameter name",
        / reason / "must be a positive integer"
      ],
      [
        / param / "second parameter name"
      ]
    ],
    / supportedFeatures / 2: "d34db33f"
  }
}
```

Figure 3: Example Extension with URI key

Obviously, an SDO like 3GPP can also easily register such a custom problem detail entry to receive a more efficient unsigned integer key; Figure 4 shows how the same example would look like using a

(made-up) registered unsigned int as custom-problem-detail-entries key:

```
{
  / title /         -1: "title of the error",
  / detail /        -2: "detailed information about the error",
  / instance /      -3: "coaps://pd.example/FA317434",
  / response-code / -4: 128, / 4.00 /

  /4711 is made-up example key that is not actually registered:/
  4711: {
    / cause /  0: "machine-readable error cause",
    / invalidParams / 1: [
      [
        / param / "first parameter name",
        / reason / "must be a positive integer"
      ],
      [
        / param / "second parameter name"
      ]
    ],
    / supportedFeatures / 2: "d34db33f"
  }
}
```

Figure 4: Example Extension with unsigned int (registered) key

In summary, the keys for the maps used inside Custom Problem Detail entries are defined specifically to the identifier of that Custom Problem Detail entry, the documentation of which defines these internal entries, typically chosen to address a given application domain.

When there is a need to evolve a Custom Problem Detail entry definition, the "ignore-unknown rule" discussed in the introduction to Section 3 provides an easy way to include additional information. The assumption is that this is done in a backward and forward compatible way. Sometimes, Custom Problem Detail entries may need to evolve in a way where forward compatibility by applying the "ignore-unknown rule" would not be appropriate: e.g., when adding a "must-understand" member, which can only be ignored at the peril of misunderstanding the Concise Problem Details data item ("false interoperability"). In this case, a new Custom Problem Detail key can simply be registered for this case, keeping the old key backward and forward compatible.

4. Privacy Considerations

Problem details may unintentionally disclose information. This can lead to both privacy and security problems. See Section 5 for more details that apply to both domains; particular attention needs to be given to unintentionally disclosing Personally Identifiable Information (PII).

5. Security Considerations

Concise Problem Details can contain URIs that are not intended to be dereferenced (Section 3.2, Paragraph 5). One reason is that dereferencing these can lead to information disclosure (tracking). Information disclosure can also be caused by URIs in problem details that *are* intended for dereferencing, e.g., the "instance" URI. Implementations need to consider which component of a client should perform the dereferencing, and which servers are trusted with serving them. In any case, the security considerations of Section 7 of [STD66] apply.

The security and privacy considerations outlined in Section 5 of [RFC7807] apply in full. While these are phrased in terms of security considerations for new RFC 7807 problem types, they equally apply to the problem detail entry definitions used here Section 3; in summary: both when defining new detail entries, and when actually generating a Concise Problem Details data item, care needs to be taken that they do not leak sensitive information. Entities storing or forwarding Concise Problem Details data items need to consider whether this leads to information being transferred out of the context within which access to sensitive information was acceptable. See also Section 3, Paragraph 6 (the last paragraph of the introduction to that section). Privacy-sensitive information in the problem details **SHOULD NOT** be obscured in ways that might lead to misclassification as non-sensitive (e.g., by base64-encoding).

6. IANA Considerations

RFC Editor: please replace RFC XXXX with the RFC number of this RFC and remove this note.

6.1. Standard Problem Detail Key registry

This specification defines a new sub-registry for Standard Problem Detail Keys in the CoRE Parameters registry [IANA.core-parameters], with the policy "specification required" (Section 4.6 of [RFC8126]).

Each entry in the registry must include:

Key value:
: a negative integer to be used as the value of the key

Name:
a name that could be used in implementations for the key

CDDL type:
type of the data associated with the key in CDDL notation

Brief description:
a brief description

Change Controller:
(see Section 2.3 of [RFC8126])

Reference:
a reference document

The expert is requested to assign the shortest key values (1+0 and 1+1 encoding) to registrations that are likely to enjoy wide use and can benefit from short encodings.

To be immediately useful in CDDL and programming language contexts, a name consists of a lower-case ASCII letter (a-z) and zero or more additional ASCII characters that are either lower-case letters, digits, or a hyphen-minus, i.e., it matches [a-z][-a-z0-9]*. As with the key values, names need to be unique.

The specification in the reference document needs to provide a description of the Standard Problem Detail entry, replicating the CDDL description in "CDDL type", and describing the semantics of the presence of this entry and the semantics of the value given with it.

Initial entries in this sub-registry are as follows:

Key value	Name	CDDL Type	Brief description	Reference
-1	title	text / tag38	short, human-readable summary of the problem shape	RFC XXXX
-2	detail	text / tag38	human-readable explanation specific to this occurrence of the problem	RFC XXXX
-3	instance	~uri	URI reference identifying specific occurrence of the problem	RFC XXXX

Key value	Name	CDDL Type	Brief description	Reference
-4	response-code	uint .size 1	CoAP response code	RFC XXXX
-5	base-uri	~uri	Base URI	RFC XXXX
-6	base-lang	tag38-ltag	Base language tag (see Appendix A)	RFC XXXX
-7	base-rtl	tag38-direction	Base writing direction (see Appendix A)	RFC XXXX
TBD	unprocessed-coap-option	one-or-more<uint>	Option number(s) of CoAP option(s) that were not understood	RFC XXXX, Section 3.1.1

Table 1: Initial Entries in the Standard Problem Detail Key registry

6.2. Custom Problem Detail Key registry

This specification defines a new sub-registry for Custom Problem Detail Keys in the CoRE Parameters registry [IANA.core-parameters], with the policy "expert review" (Section 4.5 of [RFC8126]).

The expert is instructed to attempt making the registration experience as close to first-come-first-served as reasonably achievable, but checking that the reference document does provide a description as set out below. (This requirement is a relaxed version of "specification required" as defined in Section 4.6 of [RFC8126].)

Each entry in the registry must include:

Key value:
an unsigned integer to be used as the value of the key

Name:
a name that could be used in implementations for the key

Brief description:
a brief description

Change Controller:
(see Section 2.3 of [RFC8126])

Reference:
a reference document that provides a description of the map, including a CDDL description, that describes all inside keys and values

The expert is requested to assign the shortest key values (1+0 and 1+1 encoding) to registrations that are likely to enjoy wide use and can benefit from short encodings.

To be immediately useful in CDDL and programming language contexts, a name consists of a lower-case ASCII letter (a-z) and zero or more additional ASCII characters that are either lower-case letters, digits, or a hyphen-minus, i.e., it matches [a-z][-a-z0-9]*. As with the key values, names need to be unique.

Initial entries in this sub-registry are as follows:

Key value	Name	Brief description	Reference
7807	tunnel-7807	Carry RFC 7807 problem details in a Concise Problem Details data item	RFC XXXX, Appendix B

Table 2: Initial Entries in the Custom Problem Detail Key registry

6.3. Media Type

IANA is requested to add the following Media-Type to the "Media Types" registry [IANA.media-types].

Name	Template	Reference
concise-problem-details+cbor	application/concise-problem-details+cbor	RFC XXXX, Section 6.3

Table 3: New Media Type application/concise-problem-details+cbor

Type name: application
Subtype name: concise-problem-details+cbor
Required parameters: N/A
Optional parameters: N/A
Encoding considerations: binary (CBOR data item)
Security considerations: Section 5 of RFC XXXX
Interoperability considerations: none
Published specification: Section 6.3 of RFC XXXX
Applications that use this media type: Clients and servers in the Internet of Things
Fragment identifier considerations: The syntax and semantics of fragment identifiers is as specified for "application/cbor". (At publication of RFC XXXX, there is no fragment identification syntax defined for "application/cbor".)
Person & email address to contact for further information: CoRE WG mailing list (core@ietf.org), or IETF Applications and Real-Time Area (art@ietf.org)
Intended usage: COMMON
Restrictions on usage: none

Author/Change controller: IETF
Provisional registration: no

6.4. Content-Format

IANA is requested to register a Content-Format number in the "CoAP Content-Formats" sub-registry, within the "Constrained RESTful Environments (CoRE) Parameters" Registry [IANA.core-parameters], as follows:

Content-Type	Content Coding	ID	Reference
application/concise-problem-details+cbor	-	TBD1	RFC XXXX

Table 4: New Content-Format

TBD1 is to be assigned from the space 256..999.

In the registry as defined by Section 12.3 of [RFC7252] at the time of writing, the column "Content-Type" is called "Media type" and the column "Content Coding" is called "Encoding". This paragraph to be removed by RFC editor.

6.5. CBOR Tag 38

In the registry "CBOR Tags" [IANA.cbor-tags], IANA has registered CBOR Tag 38. IANA is requested to replace the reference for this registration with Appendix A, RFC XXXX.

7. References

7.1. Normative References

[IANA.cbor-tags] IANA, "Concise Binary Object Representation (CBOR) Tags", 19 September 2013, <https://www.iana.org/assignments/cbor-tags>.

[IANA.core-parameters] IANA, "Constrained RESTful Environments (CoRE) Parameters", 8 June 2012, <https://www.iana.org/assignments/core-parameters>.

[IANA.media-types] IANA, "Provisional Standard Media Type Registry", 20 July 2012, <https://www.iana.org/assignments/provisional-standard-media-types>.

[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>.

[RFC4647] Phillips, A., Ed. and M. Davis, Ed., "Matching of Language Tags", BCP 47, RFC 4647, DOI 10.17487/RFC4647, September 2006, <https://www.rfc-editor.org/info/rfc4647>.

[RFC5646] Phillips, A., Ed. and M. Davis, Ed., "Tags for Identifying Languages", BCP 47, RFC 5646, DOI 10.17487/RFC5646, September 2009, <https://www.rfc-editor.org/info/rfc5646>.

[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>.

[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>.

[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>.

[RFC9165] Bormann, C., "Additional Control Operators for the Concise Data Definition Language (CDDL)", RFC 9165, DOI 10.17487/RFC9165, December 2021, <https://www.rfc-editor.org/info/rfc9165>.

[STD66] 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>.

[STD94] Bormann, C. and P. Hoffman, "Concise Binary Object Representation (CBOR)", STD 94, RFC 8949, DOI 10.17487/RFC8949, December 2020, <https://www.rfc-editor.org/info/rfc8949>.

7.2. Informative References

[I-D.ietf-httpapi-rfc7807bis] Nottingham, M., Wilde, E., and S. Dalal, "Problem Details for HTTP APIs", Work in Progress, Internet-Draft, draft-ietf-httpapi-rfc7807bis-03, 25 May 2022, <https://www.ietf.org/archive/id/draft-ietf-httpapi-rfc7807bis-03.txt>.

[RDF] Cyganiak, R., Wood, D., and M. Lanthaler, "RDF 1.1 Concepts and Abstract Syntax", W3C Recommendation, 25 February 2014, <http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/>.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

[RFC6082] Whistler, K., Adams, G., Duerst, M., Presuhn, R., Ed., and J. Klensin, "Deprecating Unicode Language Tag Characters: RFC 2482 is Historic", RFC 6082, DOI 10.17487/RFC6082, November 2010, <https://www.rfc-editor.org/info/rfc6082>.

[Unicode-14.0.0] The Unicode Consortium, "The Unicode Standard, Version 14.0.0", Mountain View: The Unicode Consortium, ISBN 978-1-936213-29-0, September 2021, <https://www.unicode.org/versions/Unicode14.0.0/>. Note that while this document references a version that was recent at the time of writing, the statements made based on this version are expected to remain valid for future versions.

[Unicode-14.0.0-bidi] The Unicode Consortium, "Unicode® Standard Annex #9 --- Unicode Bidirectional Algorithm", 27 August 2021, <https://www.unicode.org/reports/tr9/#Markup_And_Formatting>. Note that while this document references a version that was recent at the time of writing, the statements made based on this version are expected to remain valid for future versions.

Appendix A. Language-Tagged Strings

This appendix serves as the archival documentation for CBOR Tag 38, a tag for serializing language-tagged text strings in CBOR. The text of this appendix is adapted from the specification text supplied for its initial registration. It has been extended to allow supplementing the language tag by a direction indication.

A.1. Introduction

In some cases it is useful to specify the natural language of a text string. This specification defines a tag that does just that. One technology that supports language-tagged strings is the Resource Description Framework (RDF) [RDF].

A.2. Detailed Semantics

A language-tagged string in CBOR has the tag 38 and consists of an array with a length of 2 or 3.

The first element is a well-formed language tag under Best Current Practice 47 ([RFC5646] and [RFC4647]), represented as a UTF-8 text string (major type 3).

The second element is an arbitrary UTF-8 text string (major type 3). Both the language tag and the arbitrary string can optionally be annotated with CBOR tags; this is not shown in the CDDL below.

The optional third element, if present, represents a ternary value that indicates a direction, as follows:

* false: left-to-right direction ("ltr"). The text is expected to be displayed with left-to-right base direction if standalone, and isolated with left-to-right direction (as if enclosed in LRI ... PDI or equivalent, see [Unicode-14.0.0-bidi]) in the context of a longer string or text.

* true: right-to-left direction ("rtl"). The text is expected to be displayed with right-to-left base direction if standalone, and isolated with right-to-left direction (as if enclosed in RLI ... PDI or equivalent, see [Unicode-14.0.0-bidi]) in the context of a longer string or text.

* null indicates that no indication is made about the direction ("auto"), enabling an internationalization library to make an auto-detection decision such as treating the string as if enclosed in FSI ... PDI or equivalent, see [Unicode-14.0.0-bidi].

If the third element is absent, directionality context may be applying (e.g., base directionality information for an entire CBOR message or part thereof). If there is no directionality context applying, the default interpretation is the same as for null ("auto").

In CDDL:

```
tag38 = #6.38([tag38-ltag, text, ?tag38-direction])
tag38-ltag = text .regexp "[a-zA-Z]{1,8}(-[a-zA-Z0-9]{1,8})*"
tag38-direction = &(ltr: false, rtl: true, auto: null)
```

NOTE: Language tags of any combination of case are allowed. But Section 2.1.1 of [RFC5646], part of Best Current Practice 47, recommends a case combination for language tags that encoders that support tag 38 may wish to follow when generating language tags.

Data items with tag 38 that do not meet the criteria above are not valid (see Section 5.3.2 of [STD94]).

NOTE: The Unicode Standard [Unicode-14.0.0] includes a set of characters designed for tagging text (including language tagging), in the range U+E0000 to U+E007F. Although many applications, including RDF, do not disallow these characters in text strings, the Unicode Consortium has deprecated these characters and recommends annotating language via a higher-level protocol instead. See the section "Deprecated Tag Characters" in Section 23.9 of [Unicode-14.0.0], as well as [RFC6082].

A.3. Examples

Examples in this section are given in CBOR diagnostic notation first and then as a pretty-printed hexadecimal representation of the encoded item.

The following example shows how the English-language string "Hello" is represented.

```
38(["en", "Hello"])
```

```
D8 26              # tag(38)
   82              # array(2)
      62           # text(2)
         656E      # "en"
      65           # text(5)
         48656C6C6F # "Hello"
```

The following example shows how the French-language string "Bonjour" is represented.

```
38(["fr", "Bonjour"])
```

```
D8 26                   # tag(38)
   82                   # array(2)
      62                # text(2)
         6672           # "fr"
      67                # text(7)
         426F6E6A6F7572 # "Bonjour"
```

The following example shows how the Hebrew-language string "שלום" (HEBREW LETTER SHIN, HEBREW LETTER LAMED, HEBREW LETTER VAV, HEBREW LETTER FINAL MEM, U+05E9 U+05DC U+05D5 U+05DD) is represented. Note the rtl direction expressed by setting the third element in the array to "true".

```
38(["he", "שלום", true])
```

```
D8 26                     # tag(38)
   83                     # array(3)
      62                  # text(2)
         6865             # "he"
      68                  # text(8)
         D7A9D79CD795D79D # "שלום"
      F5                  # primitive(21)
```

Appendix B. Interworking with RFC 7807

On certain occasions, it will be necessary to carry ("tunnel") [RFC7807] problem details in a Concise Problem Details data item.

This appendix defines a Custom Problem Details entry for that purpose. This is assigned Custom Problem Detail key 7807 in Section 6.2. Its structure is:

```
tunnel-7807 = {
  ? &(type: 0) => ~uri
  ? &(status: 1) => 0..999
  * text => any
}
```

To carry an [RFC7807] problem details JSON object in a Concise Problem Details data item, first convert the JSON object to CBOR as

per Section 6.2 of [STD94]. Create an empty Concise Problem Details data item.

Move the values for "title", "detail", and "instance", if present, from the [RFC7807] problem details to the equivalent Standard Problem Detail entries. Create a Custom Problem Details entry with key 7807. Move the values for "type" and "status", if present, to the equivalent keys 0 and 1 of the Custom Problem Details entry. Move all remaining key/value pairs (additional members as per Section 3.2 of [RFC7807]) in the converted [RFC7807] problem details object to the Custom Problem Details map unchanged.

The inverse direction, carrying Concise Problem Details in a Problem Details JSON object requires the additional support provided by [I-D.ietf-httpapi-rfc7807bis], which is planned to create the HTTP Problem Types Registry. An HTTP Problem Type can then be registered that extracts top-level items from the Concise Problem Details data item in a similar way to the conversion described above, and which carries the rest of the Concise Problem Details data item in an additional member via base64url encoding without padding (Section 5 of [RFC4648]). Details can be defined in a separate document when the work on [I-D.ietf-httpapi-rfc7807bis] is completed.

Acknowledgments

Mark Nottingham and Erik Wilde, authors of RFC 7807. Klaus Hartke and Jaime Jiménez, co-authors of an earlier generation of this specification. Christian Amsüss, Marco Tiloca, Ari Keränen and Michael Richardson for review and comments on this document. Francesca Palombini for her review (and support) as responsible AD, and Joel Jaeggli for his OPSDIR review, both of which brought significant additional considerations to this document.

For Appendix A, John Cowan and Doug Ewell are also to be acknowledged. The content of an earlier version of this appendix was also discussed in the "apps-discuss at ietf.org" and "ltru at ietf.org" mailing lists. More recently, the authors initiated a discussion about the handling of writing direction information in conjunction with language tags. That led to discussions within the W3C Internationalization Core Working Group. The authors would like to acknowledge that cross-organization cooperation and particular contributions from John Klensin and Addison Phillips, and specific text proposals by Martin Dürst.

Contributors

Peter Occil

Email: poccil14 at gmail dot com
URI: http://peteroupc.github.io/CBOR/

Peter defined CBOR tag 38, basis of Appendix A.

Christian Amsüss

Email: christian@amsuess.com

Christian contributed what became Section 3.1.1.

Authors' Addresses

Thomas Fossati
arm

Email: thomas.fossati@arm.com

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

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