Problem Details for HTTP APIs
draft-nottingham-http-problem-05

Abstract

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

Note to Readers

   This draft should be discussed on the apps-discuss mailing list [1].

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 June 12, 2014.

Copyright Notice

Table of Contents

1. Introduction

HTTP [RFC2616] status codes are sometimes not sufficient to convey
enough information about an error to be helpful. While humans behind
Web browsers can be informed about the nature of the problem with an
HTML [W3C.REC-html401-19991224] response body, non-human consumers of
so-called "HTTP APIs" are usually not.

This specification defines simple JSON [RFC4627] and XML
[W3C.REC-xml-20081126] document formats to suit this purpose. They
are designed to be reused by HTTP 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 status code) and the finer-grained details of the problem
(using one of these formats).

For example, consider a response that indicates that the client's
account doesn't have enough credit. The 403 Forbidden status code
might be deemed most appropriate to use, as it will inform HTTP-
generic software (such as client libraries, caches and proxies) of
the general semantics of the response.

However, that doesn't give the API client enough information about
why the request was forbidden, the applicable account balance, or how
to correct the problem. If these details are included in the
response body in a machine-readable format, the client can treat it
appropriately; for example, triggering a transfer of more credit into
the account.

This specification does this by identifying a specific type of
problem (e.g., "out of credit") with a URI [RFC3986]; HTTP APIs can
do this by nominating new URIs under their control, or by reusing
existing ones.

Additionally, problems can contain other information, such as a URI
that identifies the specific occurrence of the problem (effectively
giving an identifier to the concept "The time Joe didn't have enough
credit last Thursday"), which may be useful for support or forensic
purposes.

The data model for problem details is a JSON [RFC4627] object; when
formatted as a JSON document, it uses the "application/problem+json"
media type. Appendix A defines how to express them in an equivalent
XML format, which uses the "application/problem+xml" media type.

Note that problem details are (naturally) not the only way to convey
the details of a problem in HTTP; if the response is still a

representation of a resource, for example, it's often preferable to
accommodate describing the relevant details in that application's
format. Likewise, in many situations, there is an appropriate HTTP
status code that does not require extra detail to be conveyed.

Instead, the aim of this specification is to define common error
formats for those applications that need one, so that they aren't
required to define their own, or worse, tempted to re-define the
semantics of existing HTTP status codes.

2. Requirements

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

3. The Problem Details JSON Object

The canonical model for problem details is a JSON [RFC4627] object.

When serialised as a JSON document, that format is identified with
the "application/problem+json" media type.

For example, a HTTP response carrying JSON problem details:

```
HTTP/1.1 403 Forbidden
Content-Type: application/problem+json
Content-Language: en

{
 "type": "http://example.com/probs/out-of-credit",
 "title": "You do not have enough credit.",
 "detail": "Your current balance is 30, but that costs 50.",
 "instance": "http://example.net/account/12345/msgs/abc",
 "balance": 30,
 "accounts": ["http://example.net/account/12345",
              "http://example.net/account/67890"]
}
```

Here, the out-of-credit problem (identified by its type URI)
indicates the reason for the 403 in "title", gives a reference for
the specific problem occurrence with "instance", gives occurrence-
specific details in "detail", and adds two extensions; "balance"
conveys the account's balance, and "accounts" gives a link where the
account can be topped up.

Note that "type" is case-sensitive in the JSON object, as are all
other member names.

3.1. Required Members

A problem details object MUST have the following members:

o "type" (string) - An absolute URI [RFC3986] that identifies the
   problem type. When dereferenced, it SHOULD provide human-readable
   documentation for the problem type (e.g., using HTML
   [W3C.REC-html401-19991224]).
o "title" (string) - A short, human-readable summary of the problem
   type. It SHOULD NOT change from occurrence to occurrence of the
   problem, except for purposes of localisation.

Consumers MUST use the type string as the primary identifier for the
problem type; the title string is advisory, and included only for
users who are not aware of the semantics of the URI, and don't have
the ability to discover them (e.g., offline log analysis). Consumers
SHOULD NOT automatically dereference the type URI.

3.2. Optional Members

Furthermore, a problem details object MAY have the following members:

o "status" (number) - The HTTP status code ([RFC2616], Section 6)
   generated by the origin server for this occurrence of the problem.
o "detail" (string) - An human readable explanation specific to this
   occurrence of the problem.
o "instance" (string) - An absolute URI that identifies the specific
   occurrence of the problem. It may or may not yield further
   information if dereferenced.

The status member, if present, is only advisory; it conveys the HTTP
status code used for the convenience of the consumer. Generators
MUST use the same status code in the actual HTTP response, to assure
that generic HTTP software that does not understand this format still
behaves correctly. See Section 6 for further caveats regarding its
use.

The detail member, if present, SHOULD focus on helping the client
correct the problem, rather than giving debugging information.

Consumers SHOULD NOT parse the detail member for information;
extensions are more suitable and less error-prone ways to obtain such
information.

3.3. Extension Members

Finally, problem type definitions MAY extend the problem details
object with additional members.

Clients consuming problem details MUST ignore any such extensions
that they don't recognise; this allows problem types to evolve and
include additional information in the future.

4. Defining New Problem Types

When an HTTP API needs to define a response that indicates an error
condition, it might be appropriate to do so by defining a new problem
type.

Before doing so, it's important to understand what they are good for,
and what's better left to other mechanisms.

Problem details are not a debugging tool for the underlying
implementation; rather, they are a way to expose greater detail about
the HTTP interface itself. New problem types need to carefully
consider the Security Considerations (Section 6); in particular the
risk of exposing attack vectors by exposing implementation internals
through error messages.

Likewise, truly generic problems - i.e., conditions that could
potentially apply to any resource on the Web - are usually better
expressed as plain status codes. For example, a "write access
disallowed" problem is probably unnecessary, since a 403 Forbidden
status code on a PUT request is self-explanatory.

Finally, an application may have a more appropriate way to carry an
error in a format that it already defines. Problem details are
intended to avoid the necessity of establishing new "fault" or
"error" document formats, not to replace existing domain-specific
formats.

That said, it is possible to add support for problem details to
existing HTTP APIs using HTTP content negotiation (e.g., using the
Accept request header to indicate a preference for this format).

New problem type definitions MUST document:
1. A type URI (typically, with the "http" scheme),
2. A title that appropriately describes it (think short), and
3. The HTTP status code for it to be used with.

Problem types MAY specify the use of the Retry-After response header

in appropriate circumstances.

A problem's type URI SHOULD resolve to HTML
[W3C.REC-html401-19991224] documentation that explains how to resolve
the problem.

A problem type definition MAY specify additional members on the
Problem Details object. For example, an extension might use typed
links [RFC5988] to another resource that can be used by machines to
resolve the problem.

If such additional members are defined, their names SHOULD start with
a letter (ALPHA, as per [RFC5234]) and SHOULD consist of characters
from ALPHA, DIGIT, and "_" (so that it can be serialized in formats
other than JSON), and SHOULD be three characters or longer.

4.1. Example

For example, if you are publishing an HTTP API to your online
shopping cart, you might need to indicate that the user is out of
credit (our example from above), and therefore cannot make the
purchase.

If you already have an application-specific format that can
accommodate this information, it's probably best to do that.
However, if you don't, you might consider using one of the problem
details formats; JSON if your API is JSON-based, or XML if it uses
that convention.

To do so, you might look for an already-defined type URI that suits
your purposes. If one is available, you can reuse that URI.

If one isn't available, you could mint and document a new type URI
(which ought to be under your control and stable over time), an
appropriate title and the HTTP status code that it will be used with,
along with what it means and how it should be handled.

In summary: an instance URI will always identify a specific
occurrence of a problem. On the other hand, type URIs can be reused
if an appropriate description of a problem type is already available
someplace else, or they can be created for new problem types.

4.2. Pre-Defined Problem Types

This specification reserves the use of one URI as a problem type.

The "about:blank" URI [RFC6694], when used as a problem type,
indicates that the problem has no additional semantics beyond that of

the HTTP status code.

When "about:blank" is used, the title SHOULD be the same as the recommended HTTP status phrase for that code (e.g., "Not Found" for 404, and so on), although it MAY be localized to suit client preferences (expressed with the Accept-Language request header).

5. Using Problem Details with Other Formats

In some situations, it can be advantageous to embed Problem Details in formats other than those described here. For example, an API that uses HTML [W3C.REC-html401-19991224] might want to also use HTML for expressing its problem details.

Problem details can be embedded in other formats by either encapsulating one of the existing serializations (JSON or XML) into that format, or by translating the model of a Problem Detail (as specified in Section 3) into the format's conventions.

For example, in HTML, a problem could be embedded by encapsulating JSON in a script tag:

```
    <script type="application/problem+json">
      {
       "type": "http://example.com/probs/out-of-credit",
       "title": "You do not have enough credit.",
       "detail": "Your current balance is 30, but that costs 50.",
       "instance": "http://example.net/account/12345/msgs/abc",
       "balance": 30,
       "accounts": ["http://example.net/account/12345",
                    "http://example.net/account/67890"]
      }
    </script>
}
```

or by inventing a mapping into RDFa [W3C.REC-rdfa-core-20120607].

This specification does not make specific recommendations regarding embedding Problem Details in other formats; the appropriate way to embed them depends both upon the format in use and application of that format.

6. Security Considerations

When defining a new problem type, the information included must be carefully vetted. Likewise, when actually generating a problem -

however it is serialized - the details given must also be
scrutinized.

Risks include leaking information that can be exploited to compromise
the system, access to the system, or the privacy of users of the
system.

Generators providing links to occurrence information are encouraged
to avoid making implementation details such as a stack dump available
through the HTTP interface, since this can expose sensitive details
of the server implementation, its data, and so on.

The "status" member duplicates the information available in the HTTP
status code itself, thereby bringing the possibility of disagreement
between the two. Their relative precedence is not clear, since a
disagreement might indicate that (for example) an intermediary has
modified the HTTP status code in transit. As such, those defining
problem types as well as generators and consumers of problems need to
be aware that generic software (such as proxies, load balancers,
firewalls, virus scanners) are unlikely to know of or respect the
status code conveyed in this member.

7. IANA Considerations

This specification defines two new Internet media types [RFC6838]:

    Type name: application
    Subtype name: problem+json
    Required parameters: None
    Optional parameters: None; unrecognised parameters
                         should be ignored
    Encoding considerations: Same as [RFC4627]
    Security considerations: see [this document]
    Interoperability considerations: None.
    Published specification: [this document]
    Applications that use this media type: HTTP
    Additional information:
      Magic number(s): n/a
      File extension(s): n/a
      Macintosh file type code(s): n/a
    Person & email address to contact for further information:
      Mark Nottingham <mnot@mnot.net>
    Intended usage: COMMON
    Restrictions on usage: None.
    Author: Mark Nottingham <mnot@mnot.net>
    Change controller: IESG

      Type name: application
      Subtype name: problem+xml
      Required parameters: None
      Optional parameters: None; unrecognized parameters
                           should be ignored
      Encoding considerations: Same as [RFC3023]
      Security considerations: see [this document]
      Interoperability considerations: None.
      Published specification: [this document]
      Applications that use this media type: HTTP
      Additional information:
        Magic number(s): n/a
        File extension(s): n/a
        Macintosh file type code(s): n/a
      Person & email address to contact for further information:
        Mark Nottingham <mnot@mnot.net>
      Intended usage: COMMON
      Restrictions on usage: None.
      Author: Mark Nottingham <mnot@mnot.net>
      Change controller: IESG

8. Acknowledgements

   The authors would like to thank Jan Algermissen, Mike Amundsen, Subbu
   Allamaraju, Roy Fielding, Eran Hammer, Sam Johnston, Mike McCall,
   Julian Reschke, and James Snell for review of this specification.

9. References

9.1. Normative References

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

   [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H.,
              Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext
              Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

   [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform
              Resource Identifier (URI): Generic Syntax", STD 66,
              RFC 3986, January 2005.

   [RFC4627] Crockford, D., "The application/json Media Type for
              JavaScript Object Notation (JSON)", RFC 4627, July 2006.

   [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax

Specifications: ABNF", STD 68, RFC 5234, January 2008.

9.2. Informative References

[ISO-19757-2]
          International Organization for Standardization,
          "Information Technology --- Document Schema Definition
          Languages (DSDL) --- Part 2: Grammar-based Validation ---
          RELAX NG", ISO/IEC 19757-2, 2003.

[RFC3023] Murata, M., St. Laurent, S., and D. Kohn, "XML Media
          Types", RFC 3023, January 2001.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, October 2010.

[RFC6694] Moonesamy, S., "The "about" URI Scheme", RFC 6694,
          August 2012.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type
          Specifications and Registration Procedures", BCP 13,
          RFC 6838, January 2013.

[W3C.REC-html401-19991224]
          Hors, A., Raggett, D., and I. Jacobs, "HTML 4.01
          Specification", World Wide Web Consortium
          Recommendation REC-html401-19991224, December 1999,
          <http://www.w3.org/TR/1999/REC-html401-19991224>.

[W3C.REC-rdfa-core-20120607]
          Adida, B., Birbeck, M., McCarron, S., and I. Herman, "RDFa
          Core 1.1", World Wide Web Consortium Recommendation REC-
          rdfa-core-20120607, June 2012,
          <http://www.w3.org/TR/2012/REC-rdfa-core-20120607>.

[W3C.REC-xml-20081126]
          Yergeau, F., Maler, E., Paoli, J., Sperberg-McQueen, C.,
          and T. Bray, "Extensible Markup Language (XML) 1.0 (Fifth
          Edition)", World Wide Web Consortium Recommendation REC-
          xml-20081126, November 2008,
          <http://www.w3.org/TR/2008/REC-xml-20081126>.

URIs

   [1]   <https://www.ietf.org/mailman/listinfo/apps-discuss>

Appendix A. HTTP Problems and XML

   Some HTTP-based APIs use XML [W3C.REC-xml-20081126] as their primary
   format convention. Such APIs MAY express problem details using the
   format defined in this appendix.

   The OPTIONAL RELAX NG schema [ISO-19757-2] for the XML format is:

   default namespace ns = "urn:ietf:rfc:XXXX"

   start = problem

   problem =
     element problem {
       (element type   { xsd:anyURI }
        & element title   { xsd:string }
        & element detail   { xsd:string }?
        & element status   { xsd:positiveInteger }?
        & element instance   { xsd:anyURI }?),
       anyNsElement
     }

   anyNsElement =
     (element   ns:* { anyNsElement | text }
      | attribute *   { text })*

   The media type for this format is "application/problem+xml".

   Extension arrays and objects can be serialized into the XML format by
   considering an element containing a child or children to represent an
   object, except for elements that contain only child element(s) named
   'i', which are considered arrays. For example, an alternate version
   of the example above would appear in XML as:

```
HTTP/1.1 403 Forbidden
Content-Type: application/problem+xml
Content-Language: en

<?xml version="1.0" encoding="UTF-8"?>
<problem xmlns="urn:ietf:rfc:XXXX">
  <type>http://example.com/probs/out-of-credit</type>
  <title>You do not have enough credit.</title>
  <detail>Your current balance is 30, but that costs 50.</detail>
  <instance>
    http://example.net/account/12345/msgs/abc
  </instance>
  <balance>30</balance>
  <accounts>
    <i>http://example.net/account/12345</i>
    <i>http://example.net/account/67890</i>
  </accounts>
</problem>
```

Note that this format uses an XML Namespace. This is primarily to allow embedding it into other formats; it does not imply that it can be extended with content from other namespaces. The RELAX NG schema explicitly only allows elements from the one namespace used in the XML format. Any extension arrays and objects MUST be serialized into XML markup using that namespace.

Authors' Addresses

Mark Nottingham
Akamai

Email: mnot@mnot.net
URI: http://www.mnot.net/

Erik Wilde
EMC
6801 Koll Center Parkway
Pleasanton, CA 94566
U.S.A.

Phone: +1-925-6006244
Email: erik.wilde@emc.com
URI: http://dret.net/netdret/