Network Working Group
Internet-Draft
Intended status: Informational
Expires: June 13, 2013

M. Nottingham
Akamai

December 10, 2012

Problem Details for HTTP APIs
draft-nottingham-http-problem-02

Abstract

This document defines a "problem detail" as an extensible way to carry machine-readable details of errors in a HTTP response, to avoid the need to invent new 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 13, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction

While HTTP [RFC2616] defines the status code as the primary indicator of generic response semantics, it is sometimes not fine-grained enough to convey helpful information about an error, particularly to non-human consumers of so-called "HTTP APIs".

Consider a 403 Forbidden response that indicates that the client's account doesn't have enough credit. While this can be adequately expressed in HTML if presented to a human in front of a Web browser, a non-browser client would have difficulty understanding the response, because it doesn't understand the structure of the markup.

This specification defines conventions for carrying machine-readable details of errors in a response ("problem details"), to avoid the need to invent new, application-specific response formats.

Conceptually, problem details are associated with a generic type (e.g., "out of credit"). Optionally, the specific occurrence of a problem can also be identified (e.g., "when Bob ran out of credit last Tuesday at 5:32 pm"). Both use URIs to assure global uniqueness, and provide the opportunity to fetch further information.

Problem details are specified as a JSON [RFC4627] object; when occurring in a message body, they use the "application/api-problem+json" media type. Appendix A defines how to translate problem details to an XML format, for those APIs that need it.

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 format.

Instead, the aim of this specification is to define a common error format for those applications that need one, so that they aren't required to define their own.

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 format for problem details is a JSON [RFC4627]

document, identified with the "application/api-problem+json" media type, whose root MUST be an object.

For example:

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

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

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

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

3.1. Required Members

The root object MUST have the following members:

- "describedBy" (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).
- "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 describedBy 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 URL, and don't have the ability to discover them (e.g., offline log analysis). Consumers SHOULD NOT automatically dereference the describedBy URL.

3.2. Optional Members

Furthermore, the root object MAY have the following members:

- "httpStatus" (number) - The HTTP status code set by the origin server for this occurrence of the problem.
- "detail" (string) - An human readable explanation specific to this occurance of the problem.
- "supportId" (string) - An absolute URI that identifies the specific occurrence of the problem. It may or may not yield further information if dereferenced.

The httpStatus 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 5 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 be 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 root object with additional members.

Clients consuming problem details MUST ignore unrecognised extensions; this allows problem types to evolve and include additional information in the future.

4. Defining New Problem Types

Before defining a new type of problem detail, 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 5, 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 describedBy URL (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 describedBy URL SHOULD resolve to HTML documentation that explains how to resolve the problem.

A problem type definition MAY specify additional members on the Problem Details JSON 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 conform to token [RFC2616], so that it can be serialised in formats other than JSON, and SHOULD be three characters or longer.

Likewise, problem types defining extensions SHOULD either make their values strings, or explain how to map their values to strings, so that it's possible to include them in other formats.

5. Security Considerations

When defining a new problem type, the information included must be carefully vetted. Likewise, when actually generating a problem - however it is serialised - the details given must also be scrutinised.

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 "httpStatus" 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.

6. IANA Considerations

This specification defines two new Internet media types:

```
Type name: application
Subtype name: api-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: api-problem+xml
Required parameters: None
Optional parameters: None; unrecognised 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
```

7. Acknowledgements

The author would like to thank Jan Algermissen, Mike Amundsen, Subbu Allamaraju, Roy Fielding, Sam Johnston, Mike McCall, Julian Reschke, James Snell, and Erik Wilde for early review of this specification (even if some disagree with parts of it).

8. References

8.1. 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.

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

[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.

8.2. Informative References

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

[W3C.REC-xml-20081126]
Sperberg-McQueen, C., Yergeau, F., Maler, E., Paoli, J., 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>.

[W3C.REC-xmlschema-0-20041028]
Walmsley, P. and D. Fallside, "XML Schema Part 0: Primer Second Edition", World Wide Web Consortium Recommendation REC-xmlschema-0-20041028, October 2004, <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028>.

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 HTTP Problems using the format defined in this appendix.

The OPTIONAL XML Schema [W3C.REC-xmlschema-0-20041028] for the XML format is:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="problem">
    <xs:complexType>
      <xs:sequence maxOccurs="unbounded" minOccurs="0">
        <xs:element name="describedBy" type="xs:anyURI"/>
        <xs:element name="title" type="xs:string"/>
        <xs:element name="detail" type="xs:string" minOccurs="0"/>
        <xs:element name="httpStatus" type="xs:integer"
                    minOccurs="0"/>
        <xs:element name="supportId" type="xs:anyURI" minOccurs="0"/>
        <xs:any namespace="##any"
                minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

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

Extension arrays and objects can be serialised 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:

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

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

```
        "http://example.net/account/67890"
      ]
    }
  }
```

would appear in XML as:


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

<problem>
  <describedBy>http://example.com/probs/out-of-credit</describedBy>
  <title>You do not have enough credit.</title>
  <detail>Your current balance is 30, but that costs 50.</detail>
  <supportId>http://example.net/account/12345/msgs/abc</supportId>
  <more>
    <balance>30</balance>
    <accounts>
      <i>http://example.net/account/12345</i>
      <i>http://example.net/account/67890</i>
    </accounts>
  </more>
</problem>
```

Author's Address

Mark Nottingham
Akamai

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