Internet Engineering Task Force (IETF)
Request for Comments: 7807
Category: Standards Track
ISSN: 2070-1721

M. Nottingham
Akamai
E. Wilde

March 2016

Problem Details for HTTP APIs

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 define new error response formats for HTTP APIs.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7807.

Table of Contents

1. Introduction

HTTP [RFC7230] 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-html5-20141028] response body, non-human consumers of so-called "HTTP APIs" are usually not.

This specification defines simple JSON [RFC7159] 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, problem details 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 can be useful for support or forensic purposes.

The data model for problem details is a JSON [RFC7159] 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 redefine the semantics of existing HTTP status codes. Even if an application chooses not to use it to convey errors, reviewing its design can help guide the design decisions faced when conveying errors in an existing format.

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 [RFC7159] object.

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

For example, an HTTP response carrying JSON problem details:

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

{
 "type": "https://example.com/probs/out-of-credit",
 "title": "You do not have enough credit.",
 "detail": "Your current balance is 30, but that costs 50.",
 "instance": "/account/12345/msgs/abc",
 "balance": 30,
 "accounts": ["/account/12345",
              "/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 links where the account can be topped up.

The ability to convey problem-specific extensions allows more than one problem to be conveyed. For example:

```
HTTP/1.1 400 Bad Request
Content-Type: application/problem+json
Content-Language: en

{
"type": "https://example.net/validation-error",
"title": "Your request parameters didn't validate.",
"invalid-params": [ {
                      "name": "age",
                      "reason": "must be a positive integer"
                    },
                    {
                      "name": "color",
                      "reason": "must be 'green', 'red' or 'blue'"}
                  ]
}
```

Note that this requires each of the subproblems to be similar enough to use the same HTTP status code. If they do not, the 207 (Multi-Status) [RFC4918] code could be used to encapsulate multiple status messages.

3.1. Members of a Problem Details Object

A problem details object can have the following members:

- "type" (string) - A URI reference [RFC3986] that identifies the problem type. This specification encourages that, when dereferenced, it provide human-readable documentation for the problem type (e.g., using HTML [W3C.REC-html5-20141028]). When this member is not present, its value is assumed to be "about:blank".

- "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 localization (e.g., using proactive content negotiation; see [RFC7231], Section 3.4).

- "status" (number) - The HTTP status code ([RFC7231], Section 6) generated by the origin server for this occurrence of the problem.

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

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

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 do not have the ability to discover them (e.g., offline log analysis). Consumers SHOULD NOT automatically dereference the type URI.

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 5 for further caveats regarding its use.

Consumers can use the status member to determine what the original status 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 HTTP information. Generic HTTP software will still use the HTTP status 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 are more suitable and less error-prone ways to obtain such information.

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

3.2. Extension Members

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

For example, our "out of credit" problem above defines two such extensions -- "balance" and "accounts" to convey additional, problem-specific information.

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

Note that because extensions are effectively put into a namespace by the problem type, it is not possible to define new "standard" members without defining a new media type.

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. Designers of 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 in response to a PUT request is self-explanatory.

Finally, an application might 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; see [RFC7231], Section 5.3.2).

New problem type definitions MUST document:

1. a type URI (typically, with the "http" or "https" scheme),

2. a title that appropriately describes it (think short), and

3. the HTTP status code for it to be used with.

Problem type definitions MAY specify the use of the Retry-After response header ([RFC7231], Section 7.1.3) in appropriate circumstances.

A problem's type URI SHOULD resolve to HTML [W3C.REC-html5-20141028] 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], Appendix B.1) and SHOULD consist of characters from ALPHA, DIGIT ([RFC5234], Appendix B.1), and "_" (so that it can be serialized in formats other than JSON), and they 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 format.

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. Predefined 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).

Please note that according to how the "type" member is defined (Section 3.1), the "about:blank" URI is the default value for that member. Consequently, any problem details object not carrying an explicit "type" member implicitly uses this URI.

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 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 (e.g., by a proxy or cache).

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, and 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 [RFC6838].

6.1. application/problem+json

Type name: application

Subtype name: problem+json

Required parameters: None

Optional parameters: None; unrecognized parameters should be ignored

Encoding considerations: Same as [RFC7159]

Security considerations: see Section 5 of this document

Interoperability considerations: None

Published specification: RFC 7807 (this document)

Applications that use this media type: HTTP

Fragment identifier considerations: Same as for application/json ([RFC7159])

Additional information:

Deprecated alias names for this type: n/a

Magic number(s): n/a

File extension(s): n/a

Macintosh file type code(s): n/a

Person and 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

6.2. application/problem+xml

Type name: application

Subtype name: problem+xml

Required parameters: None

Optional parameters: None; unrecognized parameters should be ignored

Encoding considerations: Same as [RFC7303]

Security considerations: see Section 5 of this document

Interoperability considerations: None

Published specification: RFC 7807 (this document)

Applications that use this media type: HTTP

Fragment identifier considerations: Same as for application/xml (as specified by Section 5 of [RFC7303])

Additional information:

Deprecated alias names for this type: n/a

Magic number(s): n/a

File extension(s): n/a

Macintosh file type code(s): n/a

Person and 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. References

7.1. 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, <http://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, <http://www.rfc-editor.org/info/rfc3986>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[RFC7231] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, DOI 10.17487/RFC7231, June 2014, <http://www.rfc-editor.org/info/rfc7231>.

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

7.2. 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.

[RFC4918] Dusseault, L., Ed., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, DOI 10.17487/RFC4918, June 2007, <http://www.rfc-editor.org/info/rfc4918>.

[RFC5988] Nottingham, M., "Web Linking", RFC 5988, DOI 10.17487/RFC5988, October 2010, <http://www.rfc-editor.org/info/rfc5988>.

[RFC6694] Moonesamy, S., Ed., "The "about" URI Scheme", RFC 6694, DOI 10.17487/RFC6694, August 2012, <http://www.rfc-editor.org/info/rfc6694>.

[RFC6838] Freed, N., Klensin, J., and T. Hansen, "Media Type Specifications and Registration Procedures", BCP 13, RFC 6838, DOI 10.17487/RFC6838, January 2013, <http://www.rfc-editor.org/info/rfc6838>.

[RFC7303] Thompson, H. and C. Lilley, "XML Media Types", RFC 7303, DOI 10.17487/RFC7303, July 2014, <http://www.rfc-editor.org/info/rfc7303>.

[W3C.REC-html5-20141028] Hickson, I., Berjon, R., Faulkner, S., Leithead, T., Navara, E., O'Connor, E., and S. Pfeiffer, "HTML5", W3C Recommendation REC-html5-20141028, October 2014, <http://www.w3.org/TR/2014/REC-html5-20141028>.

[W3C.REC-rdfa-core-20130822]
Adida, B., Birbeck, M., McCarron, S., and I. Herman, "RDFa Core 1.1 - Second Edition", W3C Recommendation REC-rdfa-core-20130822, August 2013, <http://www.w3.org/TR/2013/REC-rdfa-core-20130822>.

[W3C.REC-xml-stylesheet-20101028]
Clark, J., Pieters, S., and H. Thompson, "Associating Style Sheets with XML documents 1.0 (Second Edition)", W3C Recommendation REC-xml-stylesheet-20101028, October 2010, <http://www.w3.org/TR/2010/REC-xml-stylesheet-20101028>.

Appendix A. HTTP Problems and XML

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

The RELAX NG schema [ISO-19757-2] for the XML format is as follows. Keep in mind that this schema is only meant as documentation, and not as a normative schema that captures all constraints of the XML format. Also, it would be possible to use other XML schema languages to define a similar set of constraints (depending on the features of the chosen schema language).

```
default namespace ns = "urn:ietf:rfc:7807"

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 are 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, the example above appears in XML as follows:

```
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:7807">
  <type>https://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>https://example.net/account/12345/msgs/abc</instance>
  <balance>30</balance>
  <accounts>
    <i>https://example.net/account/12345</i>
    <i>https://example.net/account/67890</i>
  </accounts>
</problem>
```

Note that this format uses an XML namespace. This is primarily to allow embedding it into other XML-based formats; it does not imply that it can or should be extended with elements or attributes in 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 only that namespace.

When using the XML format, it is possible to embed an XML processing instruction in the XML that instructs clients to transform the XML, using the referenced XSLT code [W3C.REC-xml-stylesheet-20101028]. If this code is transforming the XML into (X)HTML, then it is possible to serve the XML format, and yet have clients capable of performing the transformation display human-friendly (X)HTML that is rendered and displayed at the client. Note that when using this method, it is advisable to use XSLT 1.0 in order to maximize the number of clients capable of executing the XSLT code.

Appendix B. 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-html5-20141028] might want to also use HTML for expressing its problem details.

Problem details can be embedded in other formats either by 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": "https://example.com/probs/out-of-credit",
   "title": "You do not have enough credit.",
   "detail": "Your current balance is 30, but that costs 50.",
   "instance": "/account/12345/msgs/abc",
   "balance": 30,
   "accounts": ["/account/12345",
                "/account/67890"]
  }
</script>
```

or by inventing a mapping into RDFa [W3C.REC-rdfa-core-20130822].

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.

Acknowledgements

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

Authors' Addresses

Mark Nottingham
Akamai

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

Erik Wilde

Email: erik.wilde@dret.net
URI: http://dret.net/netdret/