Network Working Group M. Nottingham
Internet-Draft May 8, 2013
Intended status: Informational
Expires: November 9, 2013

Home Documents for HTTP APIs
draft-nottingham-json-home-03

Abstract

This document proposes a "home document" format for non-browser HTTP clients.

Note to Readers

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

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 November 9, 2013.

Copyright Notice

Copyright (c) 2013 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

There is an emerging preference for non-browser Web applications (colloquially, "HTTP APIs") to use a link-driven approach to their interactions to assure loose coupling, thereby enabling extensibility and API evolution.

This is based upon experience with previous APIs that specified static URI paths (such as "http://api.example.com/v1.0/widgets/abc123/properties"), which have resulted in brittle, tight coupling between clients and servers.

Sometimes, these APIs are documented by a document format like [WADL] that is used as a design-time description; i.e., the URIs and other information they describe are "baked into" client implementations.

In contrast, a "follow your nose" API advertises the resources available to clients using link relations [RFC5988] and the formats they support using internet media types [RFC6838]. A client can then decide - at run time - which resources to interact with based upon its capabilities (as described by link relations), and the server can safely add new resources and formats without disturbing clients that are not yet aware of them.

As such, clients need to be able to discover this information quickly and efficiently use it to interact with the server. Just as with a human-targeted "home page" for a site, we can create a "home document" for a HTTP API that describes it to non-browser clients.

Of course, an HTTP API might use any format to do so; however, there are advantages to having a standard home document format. This specification suggests one for consideration, using the JSON format [RFC4627].

1.1. Notational Conventions

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].

2. JSON Home Documents

A JSON Home Document uses the format described in [RFC4627] and has the media type "application/json-home".

Its content consists of a root object with a "resources" property, whose member names are link relation types (as defined by [RFC5988]),

and values are Resource Objects, defined below.

For example:

```
GET / HTTP/1.1
Host: example.org
Accept: application/json-home

HTTP/1.1 200 OK
Content-Type: application/json-home
Cache-Control: max-age=3600
Connection: close

{
  "resources": {
    "http://example.org/rel/widgets": {
      "href": "/widgets/"
    },
    "http://example.org/rel/widget": {
      "href-template": "/widgets/{widget_id}",
      "href-vars": {
        "widget_id": "http://example.org/param/widget"
      },
      "hints": {
        "allow": ["GET", "PUT", "DELETE", "PATCH"],
        "formats": {
          "application/json": {}
        },
        "accept-patch": ["application/json-patch"],
        "accept-post": ["application/xml"],
        "accept-ranges": ["bytes"]
      }
    }
  }
}
```

Here, we have a home document that links to a resource, "/widgets/" with the relation "http://example.org/rel/widgets". It also links to an unknown number of resources with the relation type "http://example.org/rel/widget" using a URI Template [RFC6570], along with a mapping of identifiers to a variable for use in that template.

It also gives several hints about interacting with the latter "widget" resources, including the HTTP methods usable with them, the patch formats they accept, and the fact that they support partial requests [I-D.ietf-httpbis-p5-range] using the "bytes" range-specifier.

It gives no such hints about the "widgets" resource. This does not mean that it (for example) doesn't support any HTTP methods; it means that the client will need to discover this by interacting with the resource, and/or examining the documentation for its link relation type.

Effectively, this names a set of behaviors, as described by a resource object, with a link relation type. This means that several link relations might apply to a common base URL; e.g.:

```
{
  "resources": {
        "http://example.org/rel/search-by-id": {
        "href-template": "/search?id={widget}",
                "href-vars": {
                  "widget_name": "http://example.org/param/widget"
                }
        },
    "http://example.org/rel/search-by-name": {
      "href-template": "/search?name={widget_name}",
      "href-vars": {
        "widget_name": "http://example.org/param/widget_name"
      }
    }
  }
}
```

3. Resource Objects

A Resource Object links to resources of the defined type using one of two mechanisms; either a direct link (in which case there is exactly one resource of that relation type associated with the API), or a templated link, in which case there are zero to many such resources.

Direct links are indicated with an "href" property, whose value is a URI [RFC3986].

Templated links are indicated with an "href-template" property, whose value is a URI Template [RFC6570]. When "href-template" is present, the Resource Object MUST have a "href-vars" property; see "Resolving Templated Links".

Resource Objects MUST have exactly one of the "href" and "href-vars" properties.

In both forms, the links that "href" and "href-template" refer to are URI-references [RFC3986] whose base URI is that of the JSON Home

Document itself.

Resource Objects MAY also have a "hints" property, whose value is an object that uses named Resource Hints (see Section 4) as its properties.

3.1. Resolving Templated Links

A URI can be derived from a Templated Link by treating the "href-template" value as a Level 3 URI Template [RFC6570], using the "href-vars" property to fill the template.

The "href-vars" property, in turn, is an object that acts as a mapping between variable names available to the template and absolute URIs that are used as global identifiers for the semantics and syntax of those variables.

For example, given the following Resource Object:

```
  "http://example.org/rel/widget": {
    "href-template": "/widgets/{widget_id}",
    "href-vars": {
      "widget_id": "http://example.org/param/widget"
    },
    "hints": {
      "allow": ["GET", "PUT", "DELETE", "PATCH"],
      "formats": {
        "application/json": {}
      },
      "accept-patch": ["application/json-patch"],
      "accept-post": ["application/xml"],
      "accept-ranges": ["bytes"]
    }
  }
```

If you understand that "http://example.org/param/widget" is an numeric identifier for a widget (perhaps by dereferencing that URL and reading the documentation), you can then find the resource corresponding to widget number 12345 at "http://example.org/widgets/12345" (assuming that the Home Document is located at "http://example.org/").

4. Resource Hints

Resource hints allow clients to find relevant information about interacting with a resource beforehand, as a means of optimising communications, as well as advertising available behaviours (e.g., to

aid in laying out a user interface for consuming the API).

Hints are just that - they are not a "contract", and are to only be taken as advisory. The runtime behaviour of the resource always overrides hinted information.

For example, a resource might hint that the PUT method is allowed on all "widget" resources. This means that generally, the user has the ability to PUT to a particular resource, but a specific resource might reject a PUT based upon access control or other considerations. More fine-grained information might be gathered by interacting with the resource (e.g., via a GET), or by another resource "containing" it (such as a "widgets" collection) or describing it (e.g., one linked to it with a "describedby" link relation).

This specification defines a set of common hints, based upon information that's discoverable by directly interacting with resources. See Section 9.1 for information on defining new hints.

4.1. allow

- Resource Hint Name: allow
- Description: Hints the HTTP methods that the current client will be able to use to interact with the resource; equivalent to the Allow HTTP response header.
- Specification: [this document]

Content MUST be an array of strings, containing HTTP methods.

4.2. formats

- Resource Hint Name: formats
- Description: Hints the representation types that the resource produces and consumes, using the GET and PUT methods respectively, subject to the 'allow' hint.
- Specification: [this document]

Content MUST be an object, whose keys are media types, and values are objects containing Representation Hints (see Section 5).

4.3. accept-patch

- Resource Hint Name: accept-patch
- Description: Hints the PATCH [RFC5789] request formats accepted by the resource for this client; equivalent to the Accept-Patch HTTP response header.

- o Specification: [this document]

Content MUST be an array of strings, containing media types.

When this hint is present, "PATCH" SHOULD be listed in the "allow" hint.

4.4. accept-post

- o Resource Hint Name: accept-post
- o Description: Hints the POST request formats accepted by the resource for this client.
- o Specification: [this document]

Content MUST be an array of strings, containing media types.

When this hint is present, "POST" SHOULD be listed in the "allow" hint.

4.5. accept-ranges

- o Resource Hint Name: accept-ranges
- o Description: Hints the range-specifiers available to the client for this resource; equivalent to the Accept-Ranges HTTP response header [I-D.ietf-httpbis-p5-range].
- o Specification: [this document]

Content MUST be an array of strings, containing HTTP range-specifiers.

4.6. accept-prefer

- o Resource Hint Name: accept-prefer
- o Description: Hints the preferences [I-D.snell-http-prefer] supported by the resource. Note that, as per that specifications, a preference can be ignored by the server.
- o Specification: [this document]

Content MUST be an array of strings, contain preferences.

4.7. docs

- o Resource Hint Name: docs
- o Description: Hints the location for human-readable documentation for the relation type of the resource.
- o Specification: [this document]

Content MUST be a string containing an absolute-URI [RFC3986]

referring to documentation that SHOULD be in HTML format.

4.8. precondition-req

- o Resource Hint Name: precondition-req
- o Description: Hints that the resource requires state-changing requests (e.g., PUT, PATCH) to include a precondition, as per [I-D.ietf-httpbis-p4-conditional], to avoid conflicts due to concurrent updates.
- o Specification: [this document]

Content MUST be an array of strings, with possible values "etag" and "last-modified" indicating type of precondition expected.

4.9. auth-req

- o Resource Hint Name: auth-req
- o Description: Hints that the resource requires authentication using the HTTP Authentication Framework [I-D.ietf-httpbis-p7-auth].
- o Specification: [this document]

Content MUST be an array of objects, each with a "scheme" property containing a string that corresponds to a HTTP authentication scheme, and optionally a "realms" property containing an array of zero to many strings that identify protection spaces that the resource is a member of.

For example, a Resource Object might contain the following hint:

```
{
  "auth-req": [
    {
      "scheme": "Basic",
      "realms": ["private"]
    }
  ]
}
```

4.10. status

- o Resource Hint Name: status
- o Description: Hints the status of the resource.
- o Specification: [this document]

Content MUST be a string; possible values are:

- "deprecated" - indicates that use of the resource is not recommended, but it is still available.
- "gone" - indicates that the resource is no longer available; i.e., it will return a 410 Gone HTTP status code if accessed.

5. Representation Hints

TBD

6. Creating and Serving Home Documents

When making a home document available, there are a few things to keep in mind:

- A home document is best located at a memorable URI, because its URI will effectively become the URI for the API itself to clients.
- Home documents can be personalised, just as "normal" home pages can. For example, you might advertise different URIs, and/or different kinds of link relations, depending on the client's identity.
- Home documents SHOULD be assigned a freshness lifetime (e.g., "Cache-Control: max-age=3600") so that clients can cache them, to avoid having to fetch it every time the client interacts with the service.
- Custom link relation types, as well as the URIs for variables, should lead to documentation for those constructs.

6.1. Managing Change in Home Documents

The URIs used in home documents MAY change over time. However, changing them can cause issues for clients that are relying on cached home documents containing old links.

To mitigate the impact of such changes, servers SHOULD consider:

- Reducing the freshness lifetime of home documents before a link change, so that clients are less likely to refer to an "old" document.
- Regarding the "old" and "new" URIs as equally valid references for an "overlap" period.
- After that period, handling requests for the "old" URIs appropriately; e.g., with a 404 Not Found, or by redirecting the client to the new URI.

6.2. Evolving and Mixing APIs with Home Documents

Using home documents affords the opportunity to change the "shape" of the API over time, without breaking old clients.

This includes introducing new functions alongside the old ones - by adding new link relation types with corresponding resource objects - as well as adding new template variables, media types, and so on.

It's important to realise that a home document can serve more than one "API" at a time; by listing all relevant relation types, it can effectively "mix" different APIs, allowing clients to work with different resources as they see fit.

6.3. Documenting APIs that use Home Documents

Another use case for "static" API description formats like WSDL and WADL is to generate documentation for the API from them.

An API that uses the home document format correctly won't have a need to do so, provided that the link relation types and media types it uses are well-documented already.

7. Consuming Home Documents

Clients might use home documents in a variety of ways.

In the most common case - actually consuming the API - the client will scan the Resources Object for the link relation(s) that it is interested in, and then to interact with the resource(s) referred to. Resource Hints can be used to optimise communication with the client, as well as to inform as to the permissible actions (e.g., whether PUT is likely to be supported).

Note that the home document is a "living" document; it does not represent a "contract", but rather is expected to be inspected before each interaction. In particular, links from the home document MUST NOT be assumed to be valid beyond the freshness lifetime of the home document, as per HTTP's caching model [I-D.ietf-httpbis-p6-cache].

As a result, clients SHOULD cache the home document (as per [I-D.ietf-httpbis-p6-cache]), to avoid fetching it before every interaction (which would otherwise be required).

Likewise, a client encountering a 404 Not Found on a link SHOULD obtain a fresh copy of the home document, to assure that it is up-to-date.

8. Security Considerations

Clients need to exercise care when using hints. For example, a naive client might send credentials to a server that uses the auth-req hint, without checking to see if those credentials are appropriate for that server.

9. IANA Considerations

9.1. HTTP Resource Hint Registry

This specification defines the HTTP Resource Hint Registry. See Section 4 for a general description of the function of resource hints.

In particular, resource hints are generic; that is, they are potentially applicable to any resource, not specific to one application of HTTP, nor to one particular format. Generally, they ought to be information that would otherwise be discoverable by interacting with the resource.

Hint names MUST be composed of the lowercase letters (a-z), digits (0-9), underscores ("_") and hyphens ("-"), and MUST begin with a lowercase letter.

Hint content SHOULD be described in terms of JSON [RFC4627] constructs.

New hints are registered using the Expert Review process described in [RFC5226] to enforce the criteria above. Requests for registration of new resource hints are to use the following template:

- o Resource Hint Name: [hint name]
- o Description: [a short description of the hint's semantics]
- o Specification: [reference to specification document]

Initial registrations are enumerated in Section 4.

9.2. HTTP Representation Hint Registry

TBD

9.3. Media Type Registration

TBD

10. References

10.1. Normative References

[I-D.ietf-httpbis-p6-cache]
Fielding, R., Nottingham, M., and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Caching", draft-ietf-httpbis-p6-cache-22 (work in progress), February 2013.

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

[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.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

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

[RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.

10.2. Informative References

[I-D.ietf-httpbis-p4-conditional]
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", draft-ietf-httpbis-p4-conditional-22 (work in progress), February 2013.

[I-D.ietf-httpbis-p5-range]
Fielding, R., Lafon, Y., and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", draft-ietf-httpbis-p5-range-22 (work in progress), February 2013.

[I-D.ietf-httpbis-p7-auth]
Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Authentication", draft-ietf-httpbis-p7-auth-22 (work in progress), February 2013.

[I-D.snell-http-prefer]
Snell, J., "Prefer Header for HTTP", draft-snell-http-prefer-18 (work in progress), January 2013.

[MICROFORMATS]
microformats.org, "Microformats", <http://microformats.org/>.

[RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", RFC 5789, March 2010.

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

[WADL] Hadley, M. and Sun Microsystems, "Web Application Description Language", <http://www.w3.org/Submission/wadl/>.

[apps-discuss]
IETF, "IETF Apps-Discuss Mailing List", <https://www.ietf.org/mailman/listinfo/apps-discuss>.

Appendix A. Acknowledgements

Thanks to Jan Algermissen, Mike Amundsen, Bill Burke, Graham Klyne, Leif Hedstrom, Jeni Tennison, Erik Wilde and Jorge Williams for their suggestions and feedback.

Appendix B. Frequently Asked Questions

B.1. Why not Microformats?

Browser-centric Web applications use HTML as their representation format of choice. While it is possible to augment HTML for non-browser clients (using techniques like Microformats [MICROFORMATS]), a few issues become evident when doing so:

o HTML has a very forgiving syntax. While this is appropriate for browsers (especially considering that there are many million HTML authors in the world), it makes for a less-than-precise language for machines, resulting in both overhead (parsing and transmission) as well as lack of precision.

- HTML is presentation-centric, making it tempting to reformat it from time to time, to improve the "look and feel" of a page. However, doing so can cause comparatively brittle non-browser clients to lose their understanding of the content's semantics, unless very careful controls are in place.

Because of this, it's most practical to define a separate format, and JSON is easily machine-readable, precise, and has a better chance of being managed for stability.

B.2. Why doesn't the format allow references or inheritance?

Adding inheritance or references would allow more modularity in the format and make it more compact, at the cost of considerable complexity and the associated potential for errors (both in the specification and by its users).

Since good tools and compression are effective ways to achieve the same ends, this specification doesn't attempt them.

B.3. What about authentication?

In HTTP, authentication is discoverable by interacting with the resource (usually, by getting a 401 Unauthorized response status code, along with one or more challenges). While the home document could hint it, this isn't yet done, to avoid possible security complications.

B.4. What about "Faults" (i.e., errors)?

In HTTP, errors are conveyed by HTTP status codes. While this specification could (and even may) allow enumeration of possible error conditions, there's a concern that this will encourage applications to define many such "faults", leading to tight coupling between the application and its clients.

B.5. How Do I find the schema for a format?

That isn't addressed by home documents. Ultimately, it's up to the media type accepted and generated by resources to define and constrain (or not) their syntax.

B.6. How do I express complex query arguments?

Complex queries - i.e., those that exceed the expressive power of Link Templates or would require ambiguous properties of a "resources" object - aren't intended to be defined by a home document. The appropriate way to do this is with a "form" language, much as HTML

defines.

Note that it is possible to support multiple query syntaxes on the same base URL, using more than one link relation type; see the example at the start of the document.

Appendix C. **Open Issues**

The following is a list of placeholders for open issues.

- top-level object(s)
  - contact details
  - overall documentation
  - release info?
  - ToS
  - rate limiting (per-resource?)
- Resource Hints
  - indicate a POST to 201 Created pattern
  - indicate an "action" POST
  - outbound links
  - forms?
- Representation Hints
  - format profiles
  - deprecation
- Defining new top-level and resource object properties - how new ones are minted, registry, etc.
- Discovery (e.g., conneg, .well-known, etc.)
- LIMITED include function?

Author's Address

Mark Nottingham

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