

Network Working Group
Internet-Draft
Intended status: Informational
Expires: August 19, 2017

M. Nottingham
February 15, 2017

Home Documents for HTTP APIs
draft-nottingham-json-home-06

Abstract

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

Note to Readers

The issues list for this draft can be found at <https://github.com/mnot/I-D/labels/json-home> .

The most recent (often, unpublished) draft is at <https://mnot.github.io/I-D/json-home/> .

Recent changes are listed at <https://github.com/mnot/I-D/commits/gh-pages/json-home> .

For information about implementations, see <https://github.com/mnot/I-D/wiki/json-home> .

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 August 19, 2017.

Internet-Draft

Home Documents for HTTP APIs

February 2017

Copyright Notice

Copyright (c) 2017 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	3
1.1.	Notational Conventions	4
2.	API Home Documents	4
3.	API Objects	7
4.	Resource Objects	7
4.1.	Resolving Templated Links	8
5.	Resource Hints	8
5.1.	allow	9
5.2.	formats	9
5.3.	acceptPatch	9
5.4.	acceptPost	10
5.5.	acceptPut	10
5.6.	acceptRanges	10
5.7.	acceptPrefer	11
5.8.	docs	11
5.9.	preconditionRequired	11
5.10.	authSchemes	11
5.11.	status	12
6.	Security Considerations	12
7.	IANA Considerations	12
7.1.	HTTP Resource Hint Registry	12
7.2.	Media Type Registration	13
8.	References	13
8.1.	Normative References	13
8.2.	Informative References	14
Appendix A.	Acknowledgements	15

Appendix B.	Creating and Serving Home Documents	15
B.1.	Managing Change in Home Documents	15
B.2.	Evolving and Mixing APIs with Home Documents	16
Appendix C.	Consuming Home Documents	16
Appendix D.	Frequently Asked Questions	17

D.1.	Why not use (insert other service description format)? .	17
D.2.	Why doesn't the format allow references or inheritance? .	17
D.3.	What about "Faults" (i.e., errors)?	17
D.4.	How Do I find the schema for a format?	17
D.5.	How do I express complex query arguments?	17
Author's Address	18

[1.](#) Introduction

It is becoming increasingly common to use HTTP [[RFC7230](#)] for applications other than traditional Web browsing. Such "HTTP APIs" are used to integrate processes on disparate systems, make information available to machines across the Internet, and as part of the implementation of "micro-services."

By using HTTP, these applications realise a number of benefits, from message framing to caching, and well-defined semantics that are broadly understood and useful.

Often, these applications of HTTP are defined by documenting static URLs that clients need to know and servers need to implement. Any interaction outside of these bounds is uncharted territory.

For some applications, this approach brings issues, especially when the interface changes, either due to evolution, extension or drift between implementations. Furthermore, implementing more than one instance of interface can bring further issues, as different environments have different requirements.

The Web itself offers one way to address these issues, using links [[RFC3986](#)] to navigate between states. A link-driven application discovers relevant resources at run time, using a shared vocabulary of link relations [[RFC5988](#)] and internet media types [[RFC6838](#)] to support a "follow your nose" style of interaction - just as a Web browser does to navigate the Web.

A client can then decide which resources to interact with "on the fly" 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.

Doing so can provide any of a number of benefits, including:

- o Extensibility - Because new server capabilities can be expressed as link relations, new features can be layered in without introducing a new API version; clients will discover them in the home document. This promotes loose coupling between clients and servers.

- o Evolvability - Likewise, interfaces can change gradually by introducing a new link relation and/or format while still supporting the old ones.
- o Customisation - Home documents can be tailored for the client, allowing different classes of service or different client permissions to be exposed naturally.
- o Flexible deployment - Since URLs aren't baked into documentation, the server can choose what URLs to use for a given service.
- o API mixing - Likewise, more than one API can be deployed on a given server, without fear of collisions.

Whether an application ought to use links in this fashion depends on how it is deployed; generally, the most benefit will be received when multiple instances of the service are deployed, possibly with different versions, and they are consumed by clients with different capabilities. In particular, Internet Standards that use HTTP as a substrate are likely to require the attributes described above.

This document defines a "home page" format using the JSON format [[RFC7159](#)] for APIs to use as a launching point for the interactions they offer, using links. Having a well-defined format for this purpose promotes good practice and tooling.

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. API Home Documents

An API Home Document (or, interchangeably, "home document") uses the format described in [[RFC7159](#)] and has the media type "application/json-home".

Note: this media type is not final, and will change before final publication.

Its content consists of a root object with:

- o A "resources" member, whose value is an object that describes the resources associated with the API. Its member names are link relation types (as defined by [[RFC5988](#)]), and their values are Resource Objects ([Section 4](#)).

- o Optionally, a "api" member, whose value is an API Object ([Section 3](#)) that contains information about the API as a whole.

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
```

```
{
  "api": {
    "title": "Example API",
    "links": {
      "author": "mailto:api-admin@example.com",
      "describedBy": "https://example.com/api-docs/"
    }
  }
}
```

```

"resources": {
  "tag:me@example.com,2016:widgets": {
    "href": "/widgets/"
  },
  "tag:me@example.com,2016:widget": {
    "hrefTemplate": "/widgets/{widget_id}",
    "hrefVars": {
      "widget_id": "https://example.org/param/widget"
    },
    "hints": {
      "allow": ["GET", "PUT", "DELETE", "PATCH"],
      "formats": {
        "application/json": {}
      },
      "acceptPatch": ["application/json-patch+json"],
      "acceptRanges": ["bytes"]
    }
  }
}
}

```

Here, we have a home document for the API "Example API", whose author can be contacted at the e-mail address "api-admin@example.com", and whose documentation is at "https://example.com/api-docs/".

It links to a resource "/widgets/" with the relation "tag:me@example.com,2016:widgets". It also links to an unknown number of resources with the relation type "tag:me@example.com,2016: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 and POST formats they accept, and the fact that they support partial requests [RFC7233] 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": {
    "tag:me@example.com,2016:search-by-id": {
      "hrefTemplate": "/search?id={widget_id}",
      "hrefVars": {
        "widget_id": "https://example.org/param/widget_id"
      }
    },
    "tag:me@example.com,2016:search-by-name": {
      "hrefTemplate": "/search?name={widget_name}",
      "hrefVars": {
        "widget_name": "https://example.org/param/widget_name"
      }
    }
  }
}
```

Note that the examples above use both tag [[RFC4151](#)] and https [[RFC7230](#)] URIs; any URI scheme can be used to identify link relations and other artefacts in home documents. Typically, these are not links to be followed; they are only used to identify things.

[3.](#) API Objects

An API Object contains links to information about the API itself.

Two optional members are defined:

- o "title" has a string value indicating the name of the API;

- o "links" has an object value, whose member names are link relation types [[RFC5988](#)], and values are URLs [[RFC3986](#)]. The context of these links is the API home document as a whole.

No links are required to be conveyed, but APIs might benefit from setting the following:

- o author - a suitable URL (e.g., mailto: or https:) for the author(s) of the API
- o describedBy - a link to documentation for the API
- o license - a link to the legal terms for using the API

Future members of the API Object MAY be defined by specifications that update this document.

[4.](#) Resource Objects

A Resource Object links to resources of the type indicated in their name 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 "hrefTemplate" property, whose value is a URI Template [[RFC6570](#)]. When "hrefTemplate" is present, the Resource Object MUST have a "hrefVars" property; see "Resolving Templated Links".

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

In both forms, the links that "href" and "hrefTemplate" refer to are URI-references [[RFC3986](#)] whose base URI is that of the API Home Document itself.

Resource Objects MAY also have a "hints" property, whose value is an

object that uses named Resource Hints (see [Section 5](#)) as its properties.

[4.1.](#) Resolving Templated Links

A URI can be derived from a Templated Link by treating the "hrefTemplate" value as a Level 3 URI Template [[RFC6570](#)], using the "hrefVars" property to fill the template.

The "hrefVars" 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:

```
"https://example.org/rel/widget": {
  "hrefTemplate": "/widgets/{widget_id}",
  "hrefVars": {
    "widget_id": "https://example.org/param/widget"
  },
  "hints": {
    "allow": ["GET", "PUT", "DELETE", "PATCH"],
    "formats": {
      "application/json": {}
    },
    "acceptPatch": ["application/json-patch+json"],
    "acceptRanges": ["bytes"]
  }
}
```

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

[5.](#) Resource Hints

Resource hints allow clients to find relevant information about interacting with a resource beforehand, as a means of optimizing communications, as well as advertising available behaviors (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 behavior 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 7.1](#) for information on defining new hints.

[5.1.](#) allow

- o Resource Hint Name: allow
- o 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.
- o Specification: [this document]

Content MUST be an array of strings, containing HTTP methods. As per HTTP, when GET is supported, a client MAY assume that HEAD is supported.

[5.2.](#) formats

- o Resource Hint Name: formats
- o Description: Hints the representation types that the resource makes available, using the GET method.
- o Specification: [this document]

Content MUST be an object, whose keys are media types, and values are objects, currently empty.

[5.3.](#) acceptPatch

- o Resource Hint Name: accept-Patch
- o 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.

[5.4.](#) acceptPost

- o Resource Hint Name: acceptPost
- 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.

[5.5.](#) acceptPut

- o Resource Hint Name: acceptPut
- o Description: Hints the PUT 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, "PUT" SHOULD be listed in the "allow" hint.

[5.6.](#) acceptRanges

- o Resource Hint Name: acceptRanges
- o Description: Hints the range-specifiers available to the client for this resource; equivalent to the Accept-Ranges HTTP response

header [[RFC7233](#)].

- o Specification: [this document]

Content MUST be an array of strings, containing HTTP range-specifiers (typically, "bytes").

[5.7.](#) acceptPrefer

- o Resource Hint Name: acceptPrefer
- o Description: Hints the preferences [[RFC7240](#)] 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, containing preferences.

[5.8.](#) 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.

[5.9.](#) preconditionRequired

- o Resource Hint Name: preconditionRequired
- o Description: Hints that the resource requires state-changing requests (e.g., PUT, PATCH) to include a precondition, as per [[RFC7232](#)], 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.

[5.10.](#) authSchemes

- o Resource Hint Name: authSchemes
- o Description: Hints that the resource requires authentication using the HTTP Authentication Framework [[RFC7235](#)].
- 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:

```
{
  "authSchemes": [
    {
      "scheme": "Basic",
      "realms": ["private"]
    }
  ]
}
```

[5.11.](#) 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:

- o "deprecated" - indicates that use of the resource is not recommended, but it is still available.
- o "gone" - indicates that the resource is no longer available; i.e., it will return a 404 (Not Found) or 410 (Gone) HTTP status code if accessed.

[6.](#) 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.

[7.](#) IANA Considerations

[7.1.](#) HTTP Resource Hint Registry

This specification defines the HTTP Resource Hint Registry. See [Section 5](#) 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 [[RFC7159](#)] 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 5](#).

[7.2](#). Media Type Registration

TBD

[8](#). References

[8.1](#). 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>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", [BCP 26](#), [RFC 5226](#), DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.

Nottingham

Expires August 19, 2017

[Page 13]

Internet-Draft

Home Documents for HTTP APIs

February 2017

- [RFC5988] Nottingham, M., "Web Linking", [RFC 5988](#), DOI 10.17487/RFC5988, October 2010, <<http://www.rfc-editor.org/info/rfc5988>>.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", [RFC 6570](#), DOI 10.17487/RFC6570, March 2012, <<http://www.rfc-editor.org/info/rfc6570>>.
- [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>>.

- [RFC7234] Fielding, R., Ed., Nottingham, M., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Caching", [RFC 7234](#), DOI 10.17487/RFC7234, June 2014, <<http://www.rfc-editor.org/info/rfc7234>>.

8.2. Informative References

- [RFC4151] Kindberg, T. and S. Hawke, "The 'tag' URI Scheme", [RFC 4151](#), DOI 10.17487/RFC4151, October 2005, <<http://www.rfc-editor.org/info/rfc4151>>.
- [RFC5789] Dusseault, L. and J. Snell, "PATCH Method for HTTP", [RFC 5789](#), DOI 10.17487/RFC5789, March 2010, <<http://www.rfc-editor.org/info/rfc5789>>.
- [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>>.
- [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>>.
- [RFC7232] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Conditional Requests", [RFC 7232](#), DOI 10.17487/RFC7232, June 2014, <<http://www.rfc-editor.org/info/rfc7232>>.
- [RFC7233] Fielding, R., Ed., Lafon, Y., Ed., and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Range Requests", [RFC 7233](#), DOI 10.17487/RFC7233, June 2014, <<http://www.rfc-editor.org/info/rfc7233>>.

- [RFC7235] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Authentication", [RFC 7235](#), DOI 10.17487/RFC7235, June 2014, <<http://www.rfc-editor.org/info/rfc7235>>.

[RFC7240] Snell, J., "Prefer Header for HTTP", [RFC 7240](#), DOI 10.17487/RFC7240, June 2014, <<http://www.rfc-editor.org/info/rfc7240>>.

[RFC7807] Nottingham, M. and E. Wilde, "Problem Details for HTTP APIs", [RFC 7807](#), DOI 10.17487/RFC7807, March 2016, <<http://www.rfc-editor.org/info/rfc7807>>.

[Appendix A](#). Acknowledgements

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

[Appendix B](#). Creating and Serving Home Documents

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

- o A home document is best located at a memorable URI, because its URI will effectively become the URI for the API itself to clients.
- o Home documents can be personalized, 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.
- o Home documents ought to 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.
- o Custom link relation types, as well as the URIs for variables, should lead to documentation for those constructs.

[B.1](#). Managing Change in Home Documents

The URIs used in API 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 ought to consider:

- o Reducing the freshness lifetime of home documents before a link change, so that clients are less likely to refer to an "old" document.
- o Regarding the "old" and "new" URIs as equally valid references for an "overlap" period.
- o 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.

[B.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.

[Appendix C.](#) 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 optimize 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 [[RFC7234](#)].

As a result, clients ought to cache the home document (as per [[RFC7234](#)]), to avoid fetching it before every interaction (which would otherwise be required).

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

[Appendix D](#). Frequently Asked Questions

[D.1](#). Why not use (insert other service description format)?

There are a fair number of existing service description formats, including those that specialise in "RESTful" interactions. However, these formats generally are optimised for pairwise integration, or one-server-to-many-client integration, and less capable of describing protocols where both the server and client can evolve and be extended.

[D.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.

[D.3](#). 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. See [[RFC7807](#)] for further considerations.

[D.4](#). 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.

D.5. 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.

Nottingham

Expires August 19, 2017

[Page 17]

Internet-Draft

Home Documents for HTTP APIs

February 2017

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.

Author's Address

Mark Nottingham

Email: mnot@mnot.net

URI: <https://www.mnot.net/>

Nottingham

Expires August 19, 2017

[Page 18]