Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: February 6, 2015

J. Arwe
S. Speicher
IBM
E. Wilde
UC Berkeley

Aug 5, 2014

The Accept-Post HTTP Header
draft-wilde-accept-post-03

Abstract

This specification defines a new HTTP response header field Accept-Post, which indicates server support for specific media types for entity bodies in HTTP POST requests.

Note to Readers

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

Online access to all versions and files is available on github [2].

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 February 6, 2015.

Copyright Notice

Table of Contents

1. Introduction

This specification defines a new HTTP response header field Accept-Post, which indicates server support for specific media types for entity bodies in HTTP POST requests. This header field is comparable to the Accept-Patch response header field specified together with the HTTP PATCH method [RFC5789] (notice, however, that while Accept-Patch is defined to only list specific media types, Accept-Post reuses the "media range" concept of HTTP's Accept header and thus allows media type wildcards as well).

2. Terminology

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 RFC 2119 [RFC2119].

3. The Accept-Post Response Header Field

This specification introduces a new response header field Accept-Post used to specify the document formats accepted by the server in HTTP POST requests. Accept-Post SHOULD appear in the OPTIONS response for any resource that supports the use of the POST method. The presence of the Accept-Post header in response to any method is an implicit indication that POST is allowed on the resource identified by the Request-URI. The presence of a specific document format in this header indicates that this specific format is allowed on the resource identified by the Request-URI.

The syntax for Accept-Post headers, using the ABNF [RFC5234] syntax defined in Section 5.3.2 of HTTP/1.1 [RFC7231], is given by the following definition:
Accept-Post = #(media-range [accept-params])

(Please note that this ABNF differs from the one given in Section 5.3.2 of RFC 7231 [RFC7231], which includes the header field name.)

The Accept-Post header specifies a media range as defined by HTTP [RFC7231]. The media range specifies a type of representation that can be POSTed to the Request-URI.

The app:accept element is similar to the HTTP Accept request header field [RFC7231]. Media type parameters are allowed within Accept-Post, but Accept-Post has no notion of preference - "accept-params" or "q" arguments, as specified in Section 5.3.2 of [RFC7231], are not significant.

4. IANA Considerations

This specification defines a response header field for the Hypertext Transfer Protocol (HTTP) that has been registered with the Internet Assigned Numbers Authority (IANA) following the "Registration Procedures for Message Header Fields" [RFC3864].

4.1. The Accept-Post Response Header

The Accept-Post response header should be added to the permanent registry of message header fields (see [RFC3864]), taking into account the guidelines given by HTTP/1.1 [RFC7231].

Header Field Name: Accept-Post

Applicable Protocol: Hypertext Transfer Protocol (HTTP)

Status: Standard

Author/Change controller: IETF

Specification document(s): RFC XXXX

5. Examples

Accept-Post extends the way in which interaction information can be exposed in HTTP itself. The following sections contain some examples how this can be used in concrete HTTP-based services. Based on the first example of AtomPub Section 5.1, when sending a GET request to the URI of a collection, the following response could be sent, if the server decided to support Accept-Post headers:

```
HTTP/1.1 201 OK
              Date: Fri, 23 Feb 2007 21:17:11 GMT
              Content-Length: nnn
              Content-Type: application/atom+xml;type=feed
              Accept-Post: image/gif, image/jpeg, image/png
```

In this response to the GET request of a collection URI, the server indicates that this particular collection accepts new entries in the form of GIF, JPEG, or PNG images. No parameters are used, which means that there is no server-specified preference among those media types.

5.1. Atom Publishing Protocol

The Atom Publishing Protocol (AtomPub) [RFC5023] defines a model of interacting with collections and members, based on representations

using the Atom [RFC4287] syntax. AtomPub allows clients to create new collection members by using HTTP POST, with the request being sent to the collection URI. AtomPub servers can limit the media types they accept in these POST requests, and the accepted media types are listed in an "AtomPub service document".

The Accept-Post header field does allow an AtomPub server to advertise its support for specific media types in interactions with the collection resource, without the need for a client to locate the service document and interact with it. This increases the visibility of the "POST to Create" model of AtomPub, and makes it easier for clients to find out about the capabilities of a specific collection.

While the AtomPub protocol cannot be changed retroactively, this additional way of exposing interaction guidance could make it easier for clients to interact with AtomPub services that do support the Accept-Post header field. For those that do not support Accept-Post, clients would still have to rely on using the information contained in the service document (including the sometimes tricky issue of how to locate the service document for a given collection).

5.2. Linked Data Platform

The Linked Data Platform (LDP) [W3C.WD-ldp-20140311] describes a set of best practices and simple approach for a read-write Linked Data architecture, based on HTTP access to Web resources that describe their state using the RDF data model. LDP defines LDP Containers (LDPC) and LDP Resources (LDPR). Adding new LDPRs to an LDPC is done by sending an HTTP POST request to the LDPC. An LDPC can constrain the media types it is accepting for these POST requests, and should expose its support for accepted media types via Accept-Post.

In fact, the Accept-Post header was initially developed within the W3C's LDP Working Group (LDPWG), see Appendix A for acknowledgements. It was then decided that the header itself might be useful in other contexts as well, and thus should be specified in a standalone document.

5.3. Additional Information in Error Responses

If a client POSTs an unsupported POST document, it is possible for the server to use Accept-Post to indicate the supported media types. These can be specified using a 415 (Unsupported Media Type) response when the client sends a POST document format that the server does not support for the resource identified by the Request-URI. Such a response then MAY include an Accept-Post response header notify the client what POST document media types are supported.

This example applies to all resources supporting a limited set of media types for POST requests, such as the ones listed in the previous to sections. In both AtomPub and LDP, it would be possible for a server to include an Accept-Post header in a 415 response to a failed POST request, and indicate the media types that are accepted for POST requests.

6. Implementation Status

Note to RFC Editor: Please remove this section before publication.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in RFC 6982 [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to RFC 6982, "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

6.1. Eclipse Lyo

Organization: IBM developed and contributed to the Eclipse Lyo project [3].

Name: Eclipse Lyo "LDP reference implementation" [4]

Description: A very simple reference implementation for W3C Linked Data Platform (LDP) using some base Java technologies such as JAX-RS 2.0 and Apache Jena. The goals of this reference implementation is to experiment with validating the concepts in the specification and understanding what a SDK might look like to build LDP-compliant servers. Additional goal is to validate the approach for usage in OSLC4J SDK for building OSLC [5] clients and servers.

Maturity: Early prototype/alpha.

Coverage: All parts of the specification were covered for server requirements.

Licensing: Freely distributable (Eclipse Public License (EPL) [6] and Eclipse Distribution License (EDL) [7]).

Implementation Experience: Experience is only from the server perspective of generating the HTTP response header. It was trivial using JAX-RS 2.0 mechanism using a ContainerResponseFilter on all responses. More details about this approach are described in this blog post [8].

Contact: Steve Speicher <sspeiche@gmail.com>

6.2. RWW.I/O

Organization: No particular organization. The work done is part of project RWW.I/O [9].

Name: RWW.I/O - personal linked data storage.

Description: A minimal support for LDP is now included in RWW.I/O, which is a personal linked data storage space, following the structure of a Unix file system. Currently, only LDPCs are supported, since the LDPRs are always files or directories that are being managed through RESTful operations. RWW.I/O encourages the use of .meta files to semantically describe non-LD resources (e.g. images, html, js, css, etc.), and the use of .acl files for access control rules using the WAC vocabulary. Both .meta and .acl should be used per file (i.e. photo.jpg will have a .meta.photo.jpg and a .acl.photo.jpg).

Maturity: Beta until more features from LDP spec are included (if necessary).

Coverage: LDPCs on the server side, pagination and Accept-Post header. You can test LDPC support like this: curl -H "Accept: text/turtle" https://deiu.rww.io/public/?p=1 ; You can test Accept-Post header like this: curl -v -X OPTIONS -H "Accept: text/turtle" https://deiu.rww.io/public/

Licensing: MIT license. Source code is available on GitHub [10].

Implementation Experience: Implementing current LDP features in RWW.I/O was trivial. I've also decided to add the Accept-Post header to HEAD replies, as it helps to reduce the number of

requests for a client trying to discover more information about the server.

Contact: Andrei Sambra <andrei.sambra@gmail.com>

6.3. Tivoli Workload Automation

Organization: IBM [11]

Name: Tivoli Workload Automation [12]

Description: An existing scheduling product that already implements the OSLC Automation specification [13] (both client and server roles), including creation factories for Automation Requests that accept HTTP POST requests. Since OSLC Automation offers no programmatic way for clients to know which media types are supported by the server, clients are limited in practice to those required by OSLC Automation (RDF/XML), or to making optimistic requests using other RDF media types.

Maturity: Early prototype/alpha

Coverage: All parts of the specification were covered for server and client requirements.

Licensing: proprietary

Implementation Experience: Experience from the server perspective of generating the HTTP response header is that it was trivial using JAX-RS annotations to add another response header. Client parsing of the header presented no new problems, since the syntax is almost identical to the server-side processing of an Accept header.

Contact: John Arwe <johnarwe@us.ibm.com>

6.4. Jazz for Service Management

Organization: IBM [11]

Name: Jazz for Service Management Registry Services

Description: An existing component bundled with multiple existing Cloud and Smarter Infrastructure (formerly branded as Tivoli) products. It already supports multiple resource collections that use HTTP POST requests to create new member resources, e.g. "registration records". Given that clients have no existing means by which they can know which media types the server supports, and

given that Registry Services has been adding new media types over the past few months as part of its continuous delivery process, Accept-Post is a natural fit to enable looser client coupling.

Maturity: Early prototype/alpha

Coverage: All parts of the specification were covered for server requirements.

Licensing: proprietary

Implementation Experience: Experience is only from the server perspective of generating the HTTP response header. It was easy to add a new header using JAX-RS annotations.

Contact: John Arwe <johnarwe@us.ibm.com>

7. Security Considerations

The Accept-Post header may expose information that a server would prefer to not publish. In such a case, a server can simply stop exposing the header, in which case HTTP interactions would be back to the level of standard HTTP (i.e., with no indication what kind of media types a resource accepts in POST requests).

8. Open Issues

Note to RFC Editor: Please remove this section before publication.

- Accept-Post currently uses the "media range" concept of HTTP's Accept header field. An alternative would be only support fully specified media types, which is what the Accept-Patch header field is doing. This latter solution is more constrained, and fails to address some uses cases, such as AtomPub's way of exposing collection support for POST requests.

- While "accept-post" is currently defined in the "HTTP Link Hints" draft [I-D.nottingham-link-hint], it would be good to align the way in which they work. Currently, the "accept-post" of link hints allows a list of specific media types, whereas the Accept-Post header field may contain "media ranges".

9. Change Log

Note to RFC Editor: Please remove this section before publication.

9.1. From -03 to -04

- Updating references (removing RFC 2616, adding new HTTP/1.1 RFCs).

9.2. From -02 to -03

- Adding reference to RFC 5234 (ABNF).
- Updating references.
- Adding proper registration template.

9.3. From -01 to -02

- Added header field example.
- Updated author address.
- Adding more entries to the "Implementation Status" section.

9.4. From -00 to -01

- Changed ABNF for header field from RFC 2616 to HTTPbis convention (only specify the header field value grammar).
- Added implementations (all from the LDP community for now).
- Added open issue for aligning accept-post as defined by the "HTTP Link Hints" draft.

10. References

10.1. Normative References

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

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.

10.2. Informative References

[I-D.nottingham-link-hint]
Nottingham, M., "HTTP Link Hints", draft-nottingham-link-hint-00 (work in progress), June 2013.

[RFC4287] Nottingham, M., Ed. and R. Sayre, Ed., "The Atom Syndication Format", RFC 4287, December 2005.

[RFC5023] Gregorio, J. and B. de hOra, "The Atom Publishing Protocol", RFC 5023, October 2007.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

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

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, July 2013.

[RFC7231] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content", RFC 7231, June 2014.

[W3C.WD-ldp-20140311]
Speicher, S., Arwe, J., and A. Malhotra, "Linked Data Platform 1.0", World Wide Web Consortium LastCall WD-ldp-20140311, March 2014, <http://www.w3.org/TR/2014/WD-ldp-20140311>.

URIs

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

[2] <https://github.com/dret/I-D/tree/master/accept-post>

[3] <http://eclipse.org/lyo>

[4] <http://wiki.eclipse.org/Lyo/BuildLDPSample>

[5] <http://open-services.net>

[6] <http://www.eclipse.org/legal/epl-v10.html>

[7] <http://www.eclipse.org/org/documents/edl-v10.php>

[8] <http://stevespeicher.blogspot.com/2013/08/supporting-accept-post-in-jax-rs.html>

[9] <https://rww.io/>

[10] <https://github.com/deiu/rww.io>

[11] <http://www.ibm.com/>

[12] <https://www.ibm.com/developerworks/community/forums/html/topic?id=f403c299-c1c6-4da8-8b12-f3b72de54a1a>

[13] <http://open-services.net/wiki/automation/OSLC-Automation-Specification-Version-2.0/>

Appendix A. Acknowledgements

Thanks for comments and suggestions provided by Martin Duerst, Barry Leiba, Mark Nottingham, and Julian Reschke.

This work has been done in the context of the W3C Linked Data Platform Working Group (LDPWG) [W3C.WD-ldp-20140311]; thanks for comments and suggestions provided by the working group as a whole.

Authors' Addresses

John Arwe
IBM

Email: johnarwe@us.ibm.com

Steve Speicher
IBM

Email: sspeiche@us.ibm.com

Erik Wilde
UC Berkeley

Email: dret@berkeley.edu
URI: http://dret.net/netdret/