Network Working Group
Internet-Draft
Updates: 7240 (if approved)
Intended status: Standards Track
Expires: July 12, 2017

K. Murchison
CMU

January 8, 2017

Use of the Prefer Header Field in Web Distributed Authoring and Versioning (WebDAV)
draft-murchison-webdav-prefer-13

Abstract

This document defines an update to the HTTP Prefer header field to specify how it can be used by a WebDAV client to request that certain behaviors be employed by a server while constructing a response to a request. This document defines the "depth-noroot" preference.

Editorial Note (To be removed by RFC Editor before publication)

Please send comments to the Web Distributed Authoring and Versioning (WebDAV) mailing list at <mailto:w3c-dist-auth@w3.org> [1], which may be joined by sending a message with subject "subscribe" to <mailto:w3c-dist-auth-request@w3.org> [2]. This mailing list is archived at <http://lists.w3.org/Archives/Public/w3c-dist-auth/> [3].

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 July 12, 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

[RFC7240] defines the HTTP Prefer request header field and the "return=minimal" preference which indicates that a client wishes for the server to return a minimal response to a successful request, but states that what constitutes an appropriate minimal response is left solely to the discretion of the server. Section 2 of this specification defines precisely what is expected of a server when constructing minimal responses to successful WebDAV [RFC4918] requests.

[RFC7240] also defines the "return=representation" preference which indicates that a client wishes for the server to include an entity representing the current state of the resource in the response to a successful request. Section 3 of this specification makes recommendations on when this preference should be used by clients and extends its applicability to 412 (Precondition Failed) [RFC7232] responses.

Finally, Section 4 of this specification defines the "depth-noroot" preference that can be used with WebDAV methods that support the "Depth" header field.

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].

This document references XML element types in the "DAV:" [RFC4918], "urn:ietf:params:xml:ns:caldav" [RFC4791], and "urn:ietf:params:xml:ns:carddav" [RFC6352] namespaces outside of the context of an XML fragment. When doing so, the strings "DAV:", "CALDAV:", and "CARDDAV:" will be prepended to the XML element types respectively.

2. Reducing WebDAV Response Verbosity with "return=minimal"

Some payload bodies in responses to WebDAV requests, such as 207 (Multi-Status) [RFC4918] responses, can be quite verbose or even unnecessary at times. This specification defines how the Prefer request header field, in conjunction with its "return=minimal" preference, can be used by clients to reduce the verbosity of such responses by requesting that the server omit those portions of the response that can be inferred by their absence.

2.1. Minimal PROPFIND and REPORT Responses

When a PROPFIND [RFC4918] request, or a REPORT [RFC3253] request whose report type results in a 207 (Multi-Status) response, contains a Prefer header field with a preference of "return=minimal", the server SHOULD omit all DAV:propstat XML elements containing a DAV:status XML element of value 404 (Not Found) [RFC7231] from the 207 (Multi-Status) response. If the omission of such a DAV:propstat element would result in a DAV:response XML element containing zero DAV:propstat elements, the server MUST substitute one of the following in its place:

- a DAV:propstat element consisting of an empty DAV:prop element and a DAV:status element of value 200 (OK) [RFC7231]
- a DAV:status element of value 200 (OK)

The following report types are candidates that could benefit from use of the "return=minimal" preference. NOTE: This list is not intended to be normative or exhaustive.

- DAV:expand-property [RFC3253]
- DAV:acl-principal-prop-set [RFC3744]
- DAV:principal-property-search [RFC3744]
- DAV:sync-collection [RFC6578]
- CALDAV:calendar-query [RFC4791]
- CALDAV:calendar-multiget [RFC4791]
- CARDDAV:addressbook-query [RFC6352]
- CARDDAV:addressbook-multiget [RFC6352]

See Appendix B.1 and Appendix B.2 for examples.

2.2. Minimal PROPPATCH Response

When a PROPPATCH [RFC4918] request contains a Prefer header field with a preference of "return=minimal", and all instructions are processed successfully, the server SHOULD return one of the following responses rather than a 207 (Multi-Status) response:

- 204 (No Content) [RFC7231]

- 200 (OK) [RFC7231] (preferably with a zero-length message body)

See Appendix B.3 for examples.

2.3. Minimal MKCALENDAR and MKCOL Responses

Both the MKCALENDAR [RFC4791] and Extended MKCOL [RFC5689] specifications indicate that a server MAY return a message body in response to a successful request. This specification explicitly defines the intended behavior in the presence of the Prefer header field.

When a MKCALENDAR or an Extended MKCOL request contains a Prefer header field with a preference of "return=minimal", and the collection is created with all requested properties being set successfully, the server SHOULD return a 201 (Created) [RFC7231] response with an empty (zero-length) message body.

Note that the rationale for requiring that a minimal success response have an empty body is twofold:

- [RFC4791] Section 5.3.1 states: "If a response body for a successful request is included, it MUST be a CALDAV:mkcalendar-response XML element."

- [RFC5689] Section 3 states: "When an empty response body is returned with a success request status code, the client can assume that all properties were set."

See Appendix B.4 for examples.

3. Reducing WebDAV Round-Trips with "return=representation"

[RFC7240] describes the "return=representation" preference as being intended to provide a means of optimizing communication between the client and server by eliminating the need for a subsequent GET request to retrieve the current representation of the resource following a modification. This preference is equally applicable to situations where the server itself modifies a resource, and where a resource has been modified by another client.

3.1. Successful State-Changing Requests

The state-changing methods PUT [RFC7231], COPY/MOVE [RFC4918], PATCH [RFC5789], and POST [RFC5995] can be used to create or update a resource. In some instances, such as with CalDAV Scheduling [RFC6638], the created or updated resource representation may differ from the representation sent in the body of the request or referenced

by the effective request URI. In cases where the client, upon receiving a 2xx (Successful) [RFC7231] response to its state-changing request, would normally issue a subsequent GET request to retrieve the current representation of the resource, the client can instead include a Prefer header field with the "return=representation" preference in the state-changing request.

When a state-changing request contains a Prefer header field with a preference of "return=representation", and the resource is created or updated successfully, the server SHOULD include an entity representing the current state resource in the resulting 201 (Created) or 200 (OK) [RFC7231] response. In addition to coalescing the create/update and retrieve operations into a single round-trip, by returning the current representation of the resource in the response the client will know that any changes to the resource were produced by the server rather than a concurrent client, thus providing a level of atomicity to the operation.

See Appendix B.5 for examples.

3.2. Unsuccessful Conditional State-Changing Requests

Frequently, clients using a state-changing method such as those listed above will make them conditional by including either an If-Match or If-None-Match [RFC7232] header field in the request. This is done to prevent the client from accidentally overwriting a resource whose current state has been modified by another client acting in parallel. In cases where the client, upon receiving a 412 (Precondition Failed) [RFC7232] response to its conditional state-changing request, would normally issue a subsequent GET request to retrieve the current representation of the resource, the client can instead include a Prefer header field with the "return=representation" preference in the conditional state-changing request.

When a conditional state-changing request contains a Prefer header field with a preference of "return=representation", and the specified condition evaluates to false, the server SHOULD include an entity representing the current state of the resource in the resulting 412 (Precondition Failed) [RFC7232] response.

See Appendix B.6 for examples.

4. The "depth-noroot" Processing Preference

The "depth-noroot" preference indicates that the client wishes for the server to exclude the target (root) resource from processing by

the WebDAV method and only apply the WebDAV method to the target resource's subordinate resources.

depth-noroot = "depth-noroot"

This preference is only intended to be used with WebDAV methods whose definitions explicitly provide support for the Depth [RFC4918] header field. Furthermore, this preference only applies when the Depth header field has a value of "1" or "infinity" (either implicitly or explicitly).

The "depth-noroot" preference MAY be used in conjunction with the "return=minimal" preference in a single request.

See Appendix B.1 for examples.

5. Implementation Status

< RFC Editor: before publication please remove this section, the reference to [RFC7942], and any "URIs" section >

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 [RFC7942]. 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 [RFC7942], "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".

5.1. Cyrus

The open source Cyrus [4] project is a highly scalable enterprise mail system which also supports calendaring and contacts. This production level CalDAV/CardDAV implementation supports all of the preferences described in this document and successfully interoperates with the CalDAVTester, Apple Calendar and Apple Contacts, and aCal

client implementations described below. This implementation is freely distributable under a BSD style license from Computing Services at Carnegie Mellon University [5].

5.2. Calendar and Contacts Server

The open source Calendar and Contacts Server [6] project is a standards-compliant server implementing the CalDAV and CardDAV protocols. This production level implementation supports all of the preferences described in this document and successfully interoperates with the CalDAVTester and Apple Calendar and Apple Contacts client implementations described below. This implementation is freely distributable under the terms of the Apache License, Version 2.0 [7].

5.3. Bedework

Bedework [8] is an open-source enterprise calendar system that supports public, personal, and group calendaring. This production level implementation supports the "return=minimal" preference described in this document and successfully interoperates with the CalDAVTester client implementation described below. This implementation is freely distributable under the Jasig Licensing Policy [9].

5.4. DAViCal

DAViCal [10] is a server for calendar sharing using the CalDAV protocol. This production level implementation supports the "return=minimal" preference described in this document and successfully interoperates with the CalDAVTester client implementation described below. This implementation is Free Software [11] distributable under the General Public License [12].

5.5. Apple Calendar and Apple Contacts

The widely used Apple Calendar and Apple Contacts [13] clients are standards-compliant clients implementing the CalDAV and CardDAV protocols respectively. These production level implementations support the "return=minimal" preference described in this document and successfully interoperate with the Cyrus and Calendar and Contacts Server implementations described above. These client implementations are proprietary and are distributed as part of Apple's desktop operating systems.

5.6. aCal

aCal [14] is an open source calendar client for Android which uses the CalDAV standard for communication. This implementation makes some use of each of the preferences described in this document and successfully interoperates with the Cyrus server implementation described above. This implementation is freely distributable under the General Public License [15].

5.7. CalDAVTester

CalDAVTester [16] is an open source test and performance application designed to work with CalDAV and/or CardDAV servers and tests various aspects of their protocol handling as well as performance. This widely used implementation supports all of the preferences described in this document and successfully interoperates with the server implementations described above. This implementation is freely distributable under the terms of the Apache License, Version 2.0 [17].

6. Security Considerations

No new security considerations are introduced by use of the Prefer header field with WebDAV request methods, beyond those discussed in [RFC7240] and those already inherent in those methods.

7. IANA Considerations

7.1. Preference Registration

The following preference is to be added to the HTTP Preferences Registry defined in Section 5.1 of [RFC7240].

- Preference: depth-noroot
- Description: The "depth-noroot" preference indicates that the client wishes for the server to exclude the target (root) resource from processing by the WebDAV method and only apply the WebDAV method to the target resource's subordinate resources.
- Reference: RFCXXXX, Section 4
- Notes: This preference is only intended to be used with WebDAV methods whose definitions explicitly provide support for the "Depth" [RFC4918] header field. Furthermore, this preference only applies when the "Depth" header field has a value of "1" or "infinity" (either implicitly or explicitly).

7.2. Method References

The following methods are to have their references updated in the HTTP Method Registry defined in Section 8.1 of [RFC7231].

Method Name	References
MKCALENDAR	RFC4791, Section 5.3.1; RFCXXXX, Section 2.3
MKCOL	RFC4918, Section 9.3; RFC 5689, Section 3; RFCXXXX, Section 2.3
PROPFIND	RFC4918, Section 9.1; RFCXXXX, Section 2.1
PROPPATCH	RFC4918, Section 9.2; RFCXXXX, Section 2.2
REPORT	RFC3253, Section 3.6; RFCXXXX, Section 2.1

7.3. Status Code References

The following status code is to have its references updated in the HTTP Status Code Registry defined in Section 8.2 of [RFC7231].

Value	References
412	RFC7232, Section 4.2; RFCXXXX, Section 3.2

8. Acknowledgements

The author would like to thank the following individuals for contributing their ideas and support for writing this specification: Cyrus Daboo, Helge Hess, Andrew McMillan, Arnaud Quillaud, and Julian Reschke.

The author would also like to thank the Calendaring and Scheduling Consortium for advice with this specification, and for organizing interoperability testing events to help refine it.

9. References

9.1. 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>.

[RFC3253] Clemm, G., Amsden, J., Ellison, T., Kaler, C., and J. Whitehead, "Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning)", RFC 3253, DOI 10.17487/RFC3253, March 2002, <http://www.rfc-editor.org/info/rfc3253>.

[RFC4791] Daboo, C., Desruisseaux, B., and L. Dusseault, "Calendaring Extensions to WebDAV (CalDAV)", RFC 4791, DOI 10.17487/RFC4791, March 2007, <http://www.rfc-editor.org/info/rfc4791>.

[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>.

[RFC5689] Daboo, C., "Extended MKCOL for Web Distributed Authoring and Versioning (WebDAV)", RFC 5689, DOI 10.17487/RFC5689, September 2009, <http://www.rfc-editor.org/info/rfc5689>.

[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>.

[RFC5995] Reschke, J., "Using POST to Add Members to Web Distributed Authoring and Versioning (WebDAV) Collections", RFC 5995, DOI 10.17487/RFC5995, September 2010, <http://www.rfc-editor.org/info/rfc5995>.

[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>.

[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>.

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

9.2. Informative References

[MSDN.aa493854]
Microsoft Developer Network, "PROPPATCH Method", June 2006, <http://msdn.microsoft.com/en-us/library/aa493854.aspx>.

[MSDN.aa563501]
Microsoft Developer Network, "Brief Header", June 2006, <http://msdn.microsoft.com/en-us/library/aa563501.aspx>.

[MSDN.aa563950]
Microsoft Developer Network, "Depth Header", June 2006, <http://msdn.microsoft.com/en-us/library/aa563950.aspx>.

[MSDN.aa580336]
Microsoft Developer Network, "PROPFIND Method", June 2006, <http://msdn.microsoft.com/en-us/library/aa580336.aspx>.

[RFC3744] Clemm, G., Reschke, J., Sedlar, E., and J. Whitehead, "Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol", RFC 3744, DOI 10.17487/RFC3744, May 2004, <http://www.rfc-editor.org/info/rfc3744>.

[RFC6352] Daboo, C., "CardDAV: vCard Extensions to Web Distributed Authoring and Versioning (WebDAV)", RFC 6352, DOI 10.17487/RFC6352, August 2011, <http://www.rfc-editor.org/info/rfc6352>.

[RFC6578] Daboo, C. and A. Quillaud, "Collection Synchronization for Web Distributed Authoring and Versioning (WebDAV)", RFC 6578, DOI 10.17487/RFC6578, March 2012, <http://www.rfc-editor.org/info/rfc6578>.

[RFC6638] Daboo, C. and B. Desruisseaux, "Scheduling Extensions to CalDAV", RFC 6638, DOI 10.17487/RFC6638, June 2012, <http://www.rfc-editor.org/info/rfc6638>.

[RFC7942] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", BCP 205, RFC 7942, DOI 10.17487/RFC7942, July 2016, <http://www.rfc-editor.org/info/rfc7942>.

9.3. URIs

[1] http://www.cyrusimap.org/

[2] http://www.cmu.edu/computing/

[3] http://calendarserver.org/

[4] http://www.apache.org/licenses/LICENSE-2.0.html

[5] http://www.bedework.org/

[6] http://www.jasig.org/licensing

[7] http://www.davical.org/

[8] http://www.gnu.org/philosophy/open-source-misses-the-point.html

[9] http://www.gnu.org/licenses/gpl.html

[10] http://www.apple.com/macos/

[11] http://www.acal.me/

[12] http://www.gnu.org/licenses/gpl.html

[13] http://calendarserver.org/wiki/CalDAVTester

[14] http://www.apache.org/licenses/LICENSE-2.0.html

Appendix A. The Brief and Extended Depth Request Header Fields

This document is based heavily on the Brief [MSDN.aa563501] and extended Depth [MSDN.aa563950] request header fields. The behaviors described in Section 2.1 and Section 2.2 are identical to those provided by the Brief header field when used with the PROPFIND [MSDN.aa580336] and PROPPATCH [MSDN.aa493854] methods respectively. The behavior described in Section 4 is identical to that provided by the "1,noroot" [MSDN.aa563950] and "infinity,noroot" [MSDN.aa563950] Depth header field values.

Client and server implementations that already support the Brief header field can add support for the "return=minimal" preference with nominal effort.

If a server supporting the Prefer header field receives both the Brief and Prefer header fields in a request, clients can expect the server to ignore the Brief header field and only use the Prefer header field preferences.

Appendix B. Examples

B.1. PROPFIND

B.1.1. Typical PROPFIND request/response with Depth:1

This example tries to fetch one known and one unknown property from child resources.

>> Request <<

```
PROPFIND /container/ HTTP/1.1
Host: webdav.example.com
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Depth: 1

<?xml version="1.0" encoding="UTF-8"?>
<D:propfind xmlns:D="DAV:" xmlns:X="http://ns.example.com/foobar/">
  <D:prop>
    <D:resourcetype/>
    <X:foobar/>
  </D:prop>
</D:propfind>
```

>> Response <<

```
HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8"?>
<D:multistatus xmlns:D="DAV:"
               xmlns:X="http://ns.example.com/foobar/">
  <D:response>
    <D:href>/container/</D:href>
    <D:propstat>
      <D:prop>
        <D:resourcetype>
          <D:collection/>
        </D:resourcetype>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
    <D:propstat>
      <D:prop>
        <X:foobar/>
      </D:prop>
      <D:status>HTTP/1.1 404 Not Found</D:status>
    </D:propstat>
```

```
    </D:response>
    <D:response>
      <D:href>/container/work/</D:href>
      <D:propstat>
        <D:prop>
          <D:resourcetype>
            <D:collection/>
          </D:resourcetype>
        </D:prop>
        <D:status>HTTP/1.1 200 OK</D:status>
      </D:propstat>
      <D:propstat>
        <D:prop>
          <X:foobar/>
        </D:prop>
        <D:status>HTTP/1.1 404 Not Found</D:status>
      </D:propstat>
    </D:response>
    <D:response>
      <D:href>/container/home/</D:href>
      <D:propstat>
        <D:prop>
          <D:resourcetype>
            <D:collection/>
          </D:resourcetype>
        </D:prop>
        <D:status>HTTP/1.1 200 OK</D:status>
      </D:propstat>
      <D:propstat>
        <D:prop>
          <X:foobar/>
        </D:prop>
        <D:status>HTTP/1.1 404 Not Found</D:status>
      </D:propstat>
    </D:response>
    <D:response>
      <D:href>/container/foo.txt</D:href>
      <D:propstat>
        <D:prop>
          <D:resourcetype/>
        </D:prop>
        <D:status>HTTP/1.1 200 OK</D:status>
      </D:propstat>
      <D:propstat>
        <D:prop>
          <X:foobar/>
        </D:prop>
        <D:status>HTTP/1.1 404 Not Found</D:status>
```

```
      </D:propstat>
    </D:response>
  </D:multistatus>
```

B.1.2. Minimal PROPFIND request/response with Depth:1

This example tries to fetch one known and one unknown property from child resources only.

>> Request <<

```
PROPFIND /container/ HTTP/1.1
Host: webdav.example.com
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Depth: 1
Prefer: return=minimal, depth-noroot

<?xml version="1.0" encoding="UTF-8"?>
<D:propfind xmlns:D="DAV:" xmlns:X="http://ns.example.com/foobar/">
  <D:prop>
    <D:resourcetype/>
    <X:foobar/>
  </D:prop>
</D:propfind>
```

```
>> Response <<

HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Preference-Applied: return=minimal, depth-noroot

<?xml version="1.0" encoding="utf-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>/container/work/</D:href>
    <D:propstat>
      <D:prop>
        <D:resourcetype>
          <D:collection/>
        </D:resourcetype>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
  <D:response>
    <D:href>/container/home/</D:href>
    <D:propstat>
      <D:prop>
        <D:resourcetype>
          <D:collection/>
        </D:resourcetype>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
  <D:response>
    <D:href>/container/foo.txt</D:href>
    <D:propstat>
      <D:prop>
        <D:resourcetype/>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

B.1.3. Minimal PROPFIND request/response with an empty DAV:propstat element

This example tries to fetch an unknown property from a collection.

```
>> Request <<

PROPFIND /container/ HTTP/1.1
Host: webdav.example.com
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Prefer: return=minimal

<?xml version="1.0" encoding="UTF-8"?>
<D:propfind xmlns:D="DAV:" xmlns:X="http://ns.example.com/foobar/">
  <D:prop>
    <X:foobar/>
  </D:prop>
</D:propfind>

>> Response <<

HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Preference-Applied: return=minimal

<?xml version="1.0" encoding="utf-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>/container/</D:href>
    <D:propstat>
      <D:prop/>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

B.2. REPORT

B.2.1. Typical REPORT request/response

This example tries to fetch an unknown property from several resources via the DAV:expand-property [RFC3253] REPORT type.

```
   >> Request <<

   REPORT /dav/principals/ HTTP/1.1
   Host: webdav.example.com
   Content-type: text/xml; charset=utf-8
   Content-length: xxxx

   <?xml version="1.0" encoding="utf-8"?>
   <D:expand-property xmlns:D="DAV:">
     <D:property name="current-user-principal">
       <D:property name="resourcetype"/>
       <D:property name="displayname"/>
       <D:property name="foobar"
                   namespace="http://ns.example.com/foobar"/>
       <D:property name="calendar-home-set"
                   namespace="urn:ietf:params:xml:ns:caldav">
         <D:property name="resourcetype"/>
         <D:property name="foobar"
                     namespace="http://ns.example.com/foobar"/>
       </D:property>
       <D:property name="addressbook-home-set"
                   namespace="urn:ietf:params:xml:ns:carddav">
         <D:property name="resourcetype"/>
         <D:property name="foobar"
                     namespace="http://ns.example.com/foobar"/>
       </D:property>
     </D:property>
   </D:expand-property>

   >> Response <<

   HTTP/1.1 207 Multi-Status
   Content-Type: application/xml; charset=utf-8
   Content-Length: xxxx

   <?xml version="1.0" encoding="utf-8"?>
   <D:multistatus xmlns:D="DAV:"
                  xmlns:C="urn:ietf:params:xml:ns:caldav"
                  xmlns:R="urn:ietf:params:xml:ns:carddav"
                  xmlns:X="http://ns.example.com/foobar">
     <D:response>
       <D:href>/dav/principals/</D:href>
       <D:propstat>
         <D:prop>
           <D:current-user-principal>
             <D:response>
               <D:href>/dav/principals/user/ken/</D:href>
               <D:propstat>
```

```
      <D:prop>
        <D:resourcetype>
          <D:principal/>
        </D:resourcetype>
        <D:displayname>ken</D:displayname>
        <C:calendar-home-set>
          <D:response>
            <D:href>/dav/calendars/user/ken/</D:href>
            <D:propstat>
              <D:prop>
                <D:resourcetype>
                  <D:collection/>
                </D:resourcetype>
              </D:prop>
              <D:status>HTTP/1.1 200 OK</D:status>
            </D:propstat>
            <D:propstat>
              <D:prop>
                <X:foobar/>
              </D:prop>
              <D:status>HTTP/1.1 404 Not Found</D:status>
            </D:propstat>
          </D:response>
        </C:calendar-home-set>
        <R:addressbook-home-set>
          <D:response>
            <D:href>/dav/addressbooks/user/ken/</D:href>
            <D:propstat>
              <D:prop>
                <D:resourcetype>
                  <D:collection/>
                </D:resourcetype>
              </D:prop>
              <D:status>HTTP/1.1 200 OK</D:status>
            </D:propstat>
            <D:propstat>
              <D:prop>
                <X:foobar/>
              </D:prop>
              <D:status>HTTP/1.1 404 Not Found</D:status>
            </D:propstat>
          </D:response>
        </R:addressbook-home-set>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
    <D:propstat>
      <D:prop>
```

```
                  <X:foobar/>
                </D:prop>
                <D:status>HTTP/1.1 404 Not Found</D:status>
              </D:propstat>
            </D:response>
          </D:current-user-principal>
        </D:prop>
        <D:status>HTTP/1.1 200 OK</D:status>
      </D:propstat>
    </D:response>
  </D:multistatus>
```

B.2.2. Minimal REPORT request/response

This example tries to fetch an unknown property from several resources via the DAV:expand-property [RFC3253] REPORT type.

>> Request <<

```
REPORT /dav/principals/ HTTP/1.1
Host: webdav.example.com
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Prefer: return=minimal

<?xml version="1.0" encoding="utf-8"?>
<D:expand-property xmlns:D="DAV:">
  <D:property name="current-user-principal">
    <D:property name="resourcetype"/>
    <D:property name="displayname"/>
    <D:property name="foobar"
                namespace="http://ns.example.com/foobar"/>
    <D:property name="calendar-home-set"
                namespace="urn:ietf:params:xml:ns:caldav">
      <D:property name="resourcetype"/>
      <D:property name="foobar"
                  namespace="http://ns.example.com/foobar"/>
    </D:property>
    <D:property name="addressbook-home-set"
                namespace="urn:ietf:params:xml:ns:carddav">
      <D:property name="resourcetype"/>
      <D:property name="foobar"
                  namespace="http://ns.example.com/foobar"/>
    </D:property>
  </D:property>
</D:expand-property>
```

>> Response <<

```
   HTTP/1.1 207 Multi-Status
   Content-Type: application/xml; charset=utf-8
   Content-Length: xxxx
   Preference-Applied: return=minimal

   <?xml version="1.0" encoding="utf-8"?>
   <D:multistatus xmlns:D="DAV:"
                  xmlns:C="urn:ietf:params:xml:ns:caldav"
                  xmlns:R="urn:ietf:params:xml:ns:carddav"
                  xmlns:X="http://ns.example.com/foobar">
     <D:response>
       <D:href>/dav/principals/</D:href>
       <D:propstat>
         <D:prop>
           <D:current-user-principal>
             <D:response>
               <D:href>/dav/principals/user/ken/</D:href>
               <D:propstat>
                 <D:prop>
                   <D:resourcetype>
                     <D:principal/>
                   </D:resourcetype>
                   <D:displayname>ken</D:displayname>
                   <C:calendar-home-set>
                     <D:response>
                       <D:href>/dav/calendars/user/ken/</D:href>
                       <D:propstat>
                         <D:prop>
                           <D:resourcetype>
                             <D:collection/>
                           </D:resourcetype>
                         </D:prop>
                         <D:status>HTTP/1.1 200 OK</D:status>
                       </D:propstat>
                     </D:response>
                   </C:calendar-home-set>
                   <R:addressbook-home-set>
                     <D:response>
                       <D:href>/dav/addressbooks/user/ken/</D:href>
                       <D:propstat>
                         <D:prop>
                           <D:resourcetype>
                             <D:collection/>
                           </D:resourcetype>
                         </D:prop>
                         <D:status>HTTP/1.1 200 OK</D:status>
                       </D:propstat>
                     </D:response>
```

```
                  </R:addressbook-home-set>
                </D:prop>
                <D:status>HTTP/1.1 200 OK</D:status>
              </D:propstat>
            </D:response>
          </D:current-user-principal>
        </D:prop>
        <D:status>HTTP/1.1 200 OK</D:status>
      </D:propstat>
    </D:response>
  </D:multistatus>
```

B.3. PROPPATCH

B.3.1. Typical PROPPATCH request/response

```
  >> Request <<

  PROPPATCH /container/ HTTP/1.1
  Host: webdav.example.com
  Content-Type: application/xml; charset=utf-8
  Content-Length: xxxx

  <?xml version="1.0" encoding="utf-8"?>
  <D:propertyupdate xmlns:D="DAV:">
    <D:set>
      <D:prop>
        <D:displayname>My Container</D:displayname>
      </D:prop>
    </D:set>
  </D:propertyupdate>
```

```
>> Response <<

HTTP/1.1 207 Multi-Status
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>/container/</D:href>
    <D:propstat>
      <D:prop>
        <D:displayname/>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

B.3.2. Minimal PROPPATCH request/response

```
>> Request <<

PROPPATCH /container/ HTTP/1.1
Host: webdav.example.com
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Prefer: return=minimal

<?xml version="1.0" encoding="utf-8"?>
<D:propertyupdate xmlns:D="DAV:">
  <D:set>
    <D:prop>
      <D:displayname>My Container</D:displayname>
    </D:prop>
  </D:set>
</D:propertyupdate>

>> Response <<

HTTP/1.1 200 OK
Content-Length: 0
Preference-Applied: return=minimal
```

B.4. MKCOL

B.4.1. Verbose MKCOL request/response

```
>> Request <<

MKCOL /container/ HTTP/1.1
Host: webdav.example.com
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8"?>
<D:mkcol xmlns:D="DAV:">
  <D:set>
    <D:prop>
      <D:displayname>My Container</D:displayname>
    </D:prop>
  </D:set>
</D:mkcol>

>> Response <<

HTTP/1.1 201 Created
Cache-Control: no-cache
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8"?>
<D:mkcol-response xmlns:D="DAV:">
  <D:propstat>
    <D:prop>
      <D:displayname/>
    </D:prop>
    <D:status>HTTP/1.1 200 OK</D:status>
  </D:propstat>
</D:mkcol-response>
```

B.4.2. Minimal MKCOL request/response

```
>> Request <<

MKCOL /container/ HTTP/1.1
Host: webdav.example.com
Content-Type: application/xml; charset=utf-8
Content-Length: xxxx
Prefer: return=minimal

<?xml version="1.0" encoding="utf-8"?>
<D:mkcol xmlns:D="DAV:">
  <D:set>
    <D:prop>
      <D:displayname>My Container</D:displayname>
    </D:prop>
  </D:set>
</D:mkcol>

>> Response <<

HTTP/1.1 201 Created
Cache-Control: no-cache
Content-Length: 0
Preference-Applied: return=minimal
```

B.5. POST

B.5.1. Typical resource creation and retrieval via POST + GET

Note that this request is not conditional because by using the POST [RFC5995] method the client lets the server choose the resource URI, thereby guaranteeing that it will not modify an existing resource.

```
   >> Request <<

   POST /container/work;add-member/ HTTP/1.1
   Host: caldav.example.com
   Content-Type: text/calendar; charset=utf-8
   Content-Length: xxxx

   BEGIN:VCALENDAR
   VERSION:2.0
   PRODID:-//Example Corp.//CalDAV Client//EN
   BEGIN:VEVENT
   UID:CD87465FA
   SEQUENCE:0
   DTSTAMP:20120602T185254Z
   DTSTART:20120602T160000Z
   DTEND:20120602T170000Z
   TRANSP:OPAQUE
   SUMMARY:Lunch
   ORGANIZER;CN="Ken Murchison":mailto:murch@example.com
   ATTENDEE;CN="Ken Murchison";CUTYPE=INDIVIDUAL;PARTSTAT=ACCEPTED:
    mailto:murch@example.com
   ATTENDEE;CN="John Doe";CUTYPE=INDIVIDUAL;PARTSTAT
    =NEEDS-ACTION;ROLE=REQ-PARTICIPANT;RSVP=TRUE:mailto:jdoe@
    example.com
   END:VEVENT
   END:VCALENDAR

   >> Response <<

   HTTP/1.1 201 Created
   Location: /container/work/abc.ics
   Content-Length: 0
```

Note that the server did not include any validator header fields (e.g ETag) in the response, signaling that the created representation differs from the representation sent in the body of the request. The client has to send a separate GET request to retrieve the current representation:

```
   >> Request <<

   GET /container/work/abc.ics HTTP/1.1
   Host: caldav.example.com
```

```
>> Response <<

HTTP/1.1 200 OK
Content-Type: text/calendar; charset=utf-8
Content-Length: xxxx
ETag: "nahduyejc"
Schedule-Tag: "jfd84hgbcn"

BEGIN:VCALENDAR
VERSION:2.0
PRODID:-//Example Corp.//CalDAV Server//EN
BEGIN:VEVENT
UID:CD87465FA
SEQUENCE:0
DTSTAMP:20120602T185300Z
DTSTART:20120602T160000Z
DTEND:20120602T170000Z
TRANSP:OPAQUE
SUMMARY:Lunch
ORGANIZER;CN="Ken Murchison":mailto:murch@example.com
ATTENDEE;CN="Ken Murchison";CUTYPE=INDIVIDUAL;PARTSTAT=ACCEPTED:
 mailto:murch@example.com
ATTENDEE;CN="John Doe";CUTYPE=INDIVIDUAL;PARTSTAT
 =NEEDS-ACTION;ROLE=REQ-PARTICIPANT;RSVP=TRUE;SCHEDULE-STATUS=
 1.2:mailto:jdoe@example.com
END:VEVENT
END:VCALENDAR
```

B.5.2. Streamlined resource creation and retrieval via POST

Note that this request is not conditional because by using the POST [RFC5995] method the client lets the server choose the resource URI, thereby guaranteeing that it will not modify an existing resource.

```
   >> Request <<

   POST /container/work;add-member/ HTTP/1.1
   Host: caldav.example.com
   Content-Type: text/calendar; charset=utf-8
   Content-Length: xxxx
   Prefer: return=representation

   BEGIN:VCALENDAR
   VERSION:2.0
   PRODID:-//Example Corp.//CalDAV Client//EN
   BEGIN:VEVENT
   UID:CD87465FA
   SEQUENCE:0
   DTSTAMP:20120602T185254Z
   DTSTART:20120602T160000Z
   DTEND:20120602T170000Z
   TRANSP:OPAQUE
   SUMMARY:Lunch
   ORGANIZER;CN="Ken Murchison":mailto:murch@example.com
   ATTENDEE;CN="Ken Murchison";CUTYPE=INDIVIDUAL;PARTSTAT=ACCEPTED:
    mailto:murch@example.com
   ATTENDEE;CN="John Doe";CUTYPE=INDIVIDUAL;PARTSTAT
    =NEEDS-ACTION;ROLE=REQ-PARTICIPANT;RSVP=TRUE:mailto:jdoe@
    example.com
   END:VEVENT
   END:VCALENDAR
```

```
   >> Response <<

   HTTP/1.1 201 Created
   Location: /container/work/abc.ics
   Content-Type: text/calendar; charset=utf-8
   Content-Length: xxxx
   Content-Location: /container/work/abc.ics
   ETag: "nahduyejc"
   Schedule-Tag: "jfd84hgbcn"
   Preference-Applied: return=representation

   BEGIN:VCALENDAR
   VERSION:2.0
   PRODID:-//Example Corp.//CalDAV Server//EN
   BEGIN:VEVENT
   UID:CD87465FA
   SEQUENCE:0
   DTSTAMP:20120602T185300Z
   DTSTART:20120602T160000Z
   DTEND:20120602T170000Z
   TRANSP:OPAQUE
   SUMMARY:Lunch
   ORGANIZER;CN="Ken Murchison":mailto:murch@example.com
   ATTENDEE;CN="Ken Murchison";CUTYPE=INDIVIDUAL;PARTSTAT=ACCEPTED:
    mailto:murch@example.com
   ATTENDEE;CN="John Doe";CUTYPE=INDIVIDUAL;PARTSTAT
    =NEEDS-ACTION;ROLE=REQ-PARTICIPANT;RSVP=TRUE;SCHEDULE-STATUS=
    1.2:mailto:jdoe@example.com
   END:VEVENT
   END:VCALENDAR
```

B.6. PUT

B.6.1. Typical conditional resource update failure and retrieval via PUT + GET

```
   >> Request <<

   PUT /container/motd.txt HTTP/1.1
   Host: dav.example.com
   Content-Type: text/plain
   Content-Length: xxxx
   If-Match: "asd973"

   Either write something worth reading or do something worth writing.
```

```
>> Response <<

HTTP/1.1 412 Precondition Failed
Content-Length: 0
```

The resource has been modified by another user agent (ETag mismatch), therefore the client has to send a separate GET request to retrieve the current representation:

```
>> Request <<

GET /container/motd.txt HTTP/1.1
Host: dav.example.com


>> Response <<

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: xxxx
ETag: "789sdas"

An investment in knowledge pays the best interest.
```

B.6.2. Streamlined conditional resource update failure and retrieval via PUT

```
>> Request <<

PUT /container/motd.txt HTTP/1.1
Host: dav.example.com
Content-Type: text/plain
Content-Length: xxxx
If-Match: "asd973"
Prefer: return=representation

Either write something worth reading or do something worth writing.
```

```
   >> Response <<

   HTTP/1.1 412 Precondition Failed
   Content-Type: text/plain
   Content-Length: xxxx
   Content-Location: /container/motd.txt
   ETag: "789sdas"
   Preference-Applied: return=representation

   An investment in knowledge pays the best interest.
```

Appendix C. Change Log

< RFC Editor: before publication please remove this section >

C.1. Since -12

- Several editorial and formatting changes from Julian Reschke.

C.2. Since -11

- Several fixes per Gen-ART Review (Stuart Bryant):
  - Added "updates RFC7240" text to Abstract.
  - Removed "Open Issues" Section.
  - Added RFC Editor note to remove "URIs" Section.
  - Fixed typos.

C.3. Since -10

- Pared down Updates per Alexey.
- Added self-reference for 412 status code in registry.

C.4. Since -09

- Combined PROPFIND and REPORT sections
- Added several more RFCs to Updated list.
- Added list of report types that can benefit from "return=minimal".
- Changed REPORT example to use DAV:expand-property.
- Added IANA section to update HTTP Method Registry references.

- Split "return=representation" discussion into two separate sections and expanded text.
- Updated Open Issues with new questions.
- Several editorial changes from Julian Reschke.

C.5. Since -08

- Moved examples to Appendix B.
- Added reference to HTTP PATCH.
- Updated Implementation Status reference from RFC 6982 to RFC 7942.

C.6. Since -07

- No substantive changes. Refreshed due to pending expiration.

C.7. Since -06

- Updated HTTPbis and Prefer references to published RFCs.

C.8. Since -05

- Allow a minimal PROPFIND/REPORT response to contain a DAV:status element rather than an empty DAV:propstat element.
- Allow 204 (No Content) as a minimal PROPATCH success response.
- Added justification for why a minimal MKCOL/MKCALENDAR success response must have an empty body.
- Added text and an example of how "return=representation" can be employed with a conditional state-changing request and a 412 (Precondition Failed) response.
- Added a note to the POST+GET example bringing attention to the lack of a validator header field in the POST response.
- Reduced the number of inline references.
- Limited most examples to vanilla WebDAV.
- Reduced number of items in TOC.
- Removed the recommendation that the legacy Brief header functionality should be implemented.

- Added note about how a server should handle a request that contains both Brief and Prefer.

- Other editorial tweaks from Julian Reschke.

C.9. Since -04

- Added note stating where to send comments.

C.10. Since -03

- Limited "Updates" to just RFC 4918.

- Consensus from CalConnect membership that a "depth-root" option is unnecessary at this point.

- Consensus from CalConnect membership to remove Vary header field from PROPFIND and REPORT responses since these responses don't appear to be cached.

- Updated "Implementation Status" section boilerplate to RFC 6982.

- Added aCal to "Implementation Status" section.

- Added note that servers SHOULD respond with Preference-Applied when return=minimal is used with PROPFIND or REPORT.

C.11. Since -02

- Reintroduced "Updates" to header.

- Added text noting that "return=representation" provides a level of atomicity to the operation.

- Added "Implementation Status" section.

- Tweaked/corrected some examples..

- Updated HTTPbis references.

C.12. Since -01

- Removed "Updates" from header.

- Fixed some missing/incorrect references.

- Reintroduced Cache-Control:no-cache to MKCOL responses.

C.13. Since -00

- Updated to comply with draft-snell-httpprefer-18.
- Reordered "Minimal REPORT Response" and "Minimal PROPPATCH Response" sections.
- Added some explanatory text to examples.

C.14. Since CalConnect XXIV

- Updated references.
- Stated that "depth-noroot" can be used in conjuction with "return=minimal".
- Added text mentioning that "depth-noroot" is based on the MSDN "1,noroot" and "infinity,noroot" Depth header values.
- The server behavior required when "return=minimal" would result in zero DAV:propstat elements has been changed

from:

```
<?xml version="1.0" encoding="utf-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>/container/</D:href>
    <D:status>HTTP/1.1 200 OK</D:status>
  </D:response>
</D:multistatus>
```

to the slightly more verbose:

```
<?xml version="1.0" encoding="utf-8"?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>/container/</D:href>
    <D:propstat>
      <D:prop/>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

Author's Address

Kenneth Murchison
Carnegie Mellon University
5000 Forbes Avenue
Pittsburgh, PA 15213
USA

Phone: +1 412 268 1982
Email: murch@andrew.cmu.edu