Network Working Group
Request for Comments: 3648
Category: Standards Track

J. Whitehead
U.C. Santa Cruz
J. Reschke, Ed.
greenbytes

December 2003

Web Distributed Authoring and Versioning (WebDAV) Ordered Collections Protocol

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2003). All Rights Reserved.

Abstract

This specification extends the Web Distributed Authoring and Versioning (WebDAV) Protocol to support the server-side ordering of collection members. Of particular interest are orderings that are not based on property values, and so cannot be achieved using a search protocol’s ordering option and cannot be maintained automatically by the server. Protocol elements are defined to let clients specify the position in the ordering of each collection member, as well as the semantics governing the ordering.

Table of Contents

1. Notational Conventions 3
2. Introduction 3
3. Terminology 4
4. Overview of Ordered Collections 5
4.1. Additional Collection properties 6
4.1.1. DAV:ordering-type (protected). 6
5. Creating an Ordered Collection 7
5.1. Overview 7
5.2. Example: Creating an Ordered Collection. 8
6. Setting the Position of a Collection Member. 8
6.1. Overview 8
6.2. Examples: Setting the Position of a Collection Member. . 10
6.3. Examples: Renaming a member of an ordered collection . . 10
7. Changing a Collection Ordering: ORDERPATCH method. 11
7.1. Example: Changing a Collection Ordering. 13
7.2. Example: Failure of an ORDERPATCH Request. 14
8. Listing the Members of an Ordered Collection 16
8.1. Example: PROPFIND on an Ordered Collection 17
9. Relationship to versioned collections. 19
9.1. Collection Version Properties. 20
9.1.1. Additional semantics for DAV:version-controlled-binding-set (protected) . 20
9.1.2. DAV:ordering-type (protected). 20
9.2. Additional CHECKIN semantics 20
9.3. Additional CHECKOUT Semantics. 20
9.4. Additional UNCHECKOUT, UPDATE, and MERGE Semantics . . . 21
10. Capability Discovery 21
10.1. Example: Using OPTIONS for the Discovery of Support for Ordering 22
10.2. Example: Using Live Properties for the Discovery of Ordering 22
11. Security Considerations. 23
11.1. Denial of Service and DAV:ordering-type 23
12. Internationalization Considerations. 24
13. IANA Considerations. 24
14. Intellectual Property Statement. 25
15. Contributors 25
16. Acknowledgements 25
17. Normative References 26
A. Extensions to the WebDAV Document Type Definition. 27
Index. 27
Authors' Addresses 29
Full Copyright Statement 30

1. Notational Conventions

Since this document describes a set of extensions to the WebDAV Distributed Authoring Protocol [RFC2518], which is itself an extension to the HTTP/1.1 protocol, the augmented BNF used here to describe protocol elements is exactly the same as described in Section 2.1 of HTTP [RFC2616]. Since this augmented BNF uses the basic production rules provided in Section 2.2 of HTTP, these rules apply to this document as well.

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 uses XML DTD fragments as a purely notational convention. WebDAV request and response bodies can not be validated due to the specific extensibility rules defined in section 23 of [RFC2518] and due to the fact that all XML elements defined by this specification use the XML namespace name "DAV:". In particular:

1. element names use the "DAV:" namespace,

2. element ordering is irrelevant,

3. extension elements (elements not already defined as valid child elements) may be added anywhere, except where explicitly stated otherwise,

4. extension attributes (attributes not already defined as valid for this element) may be added anywhere, except where explicitly stated otherwise.

2. Introduction

This specification builds on the collection infrastructure provided by the WebDAV Distributed Authoring Protocol, adding support for the server-side ordering of collection members.

There are many scenarios in which it is useful to impose an ordering on a collection at the server, such as expressing a recommended access order, or a revision history order. The members of a collection might represent the pages of a book, which need to be presented in order if they are to make sense, or an instructor might create a collection of course readings that she wants to be displayed in the order they are to be read.

Orderings may be based on property values, but this is not always the case. The resources in the collection may not have properties that

can be used to support the desired ordering. Orderings based on properties can be obtained using a search protocol's ordering option, but orderings not based on properties cannot. These orderings generally need to be maintained by a human user.

The ordering protocol defined here focuses on support for such human-maintained orderings. Its protocol elements allow clients to specify the position of each collection member in the collection's ordering, as well as the semantics governing the order. The protocol is designed to allow additional support in the future for orderings that are maintained automatically by the server.

The remainder of this document is structured as follows: Section 3 defines terminology that will be used throughout the specification. Section 4 provides an overview of ordered collections. Section 5 describes how to create an ordered collection, and Section 6 discusses how to set a member's position in the ordering of a collection. Section 7 explains how to change a collection ordering. Section 8 discusses listing the members of an ordered collection. Section 9 discusses the impact on version-controlled collections (as defined in [RFC3253]). Section 10 describes capability discovery. Sections 11 through 13 discuss security, internationalization, and IANA considerations. The remaining sections provide supporting information.

3. Terminology

The terminology used here follows that in [RFC2518] and [RFC3253]. Definitions of the terms resource, Uniform Resource Identifier (URI), and Uniform Resource Locator (URL) are provided in [RFC2396].

Ordered Collection

A collection for which the results from a PROPFIND request are guaranteed to be in the order specified for that collection.

Unordered Collection

A collection for which the client cannot depend on the repeatability of the ordering of results from a PROPFIND request.

Client-Maintained Ordering

An ordering of collection members that is maintained on the server based on client requests specifying the position of each collection member in the ordering.

Server-Maintained Ordering

An ordering of collection members that is maintained automatically by the server, based on a client's choice of ordering semantics.

Ordering Semantics

In general, ordering semantics are the set of structures or meanings applied to the ordering of the member of a specific collection. Within this document, "ordering semantics" refers specifically to the structure specified in the DAV:ordering-type property. See Section 4.1.1 for more information on DAV:ordering-type.

This document uses the terms "precondition", "postcondition" and "protected property" as defined in [RFC3253]. Servers MUST report pre-/postcondition failures as described in section 1.6 of this document.

4. Overview of Ordered Collections

If a collection is not ordered, the client cannot depend on the repeatability of the ordering of results from a PROPFIND request. By specifying an ordering for a collection, a client requires the server to follow that ordering whenever it responds to a PROPFIND request on that collection.

Server-side orderings may be client-maintained or server-maintained. For client-maintained orderings, a client must specify the ordering position of each of the collection's members, either when the member is added to the collection (using the Position header (Section 6)) or later (using the ORDERPATCH (Section 7) method). For server-maintained orderings, the server automatically positions each of the collection's members according to the ordering semantics. This specification supports only client-maintained orderings, but is designed to allow the future extension with server-maintained orderings.

A collection that supports ordering is not required to be ordered.

If a collection is ordered, each of its internal member URIs MUST appear in the ordering exactly once, and the ordering MUST NOT include any URIs that are not internal members of the collection. The server is responsible for enforcing these constraints on orderings. The server MUST remove an internal member URI from the ordering when it is removed from the collection. Removing an internal member MUST NOT affect the ordering of the remaining

internal members. The server MUST add an internal member URI to the ordering when it is added to the collection.

Only one ordering can be attached to any collection. Multiple orderings of the same resources can be achieved by creating multiple collections referencing those resources, and attaching a different ordering to each collection.

An ordering is considered to be part of the state of a collection resource. Consequently, the ordering is the same no matter which URI is used to access the collection and is protected by locks or access control constraints on the collection.

4.1. Additional Collection properties

A DAV:allprop PROPFIND request SHOULD NOT return any of the properties defined in this document.

4.1.1. DAV:ordering-type (protected)

The DAV:ordering-type property indicates whether the collection is ordered and, if so, uniquely identifies the semantics of the ordering. It may also point to an explanation of the semantics in human and/or machine-readable form. At a minimum, this allows human users who add members to the collection to understand where to position them in the ordering. This property cannot be set using PROPPATCH. Its value can only be set by including the Ordering-Type header with a MKCOL request or by submitting an ORDERPATCH request.

Ordering types are identified by URIs that uniquely identify the semantics of the collection’s ordering. The following two URIs are predefined:

DAV:custom: The value DAV:custom indicates that the collection is ordered, but the semantics governing the ordering are not being advertised.

DAV:unordered: The value DAV:unordered indicates that the collection is not ordered. That is, the client cannot depend on the repeatability of the ordering of results from a PROPFIND request.

An ordering-aware client interacting with an ordering-unaware server (e.g., one that is implemented only according to [RFC2518]) SHOULD assume that the collection is unordered if a collection does not have the DAV:ordering-type property.

```
<!ELEMENT ordering-type (href) >
```

5. Creating an Ordered Collection

5.1. Overview

When a collection is created, the client MAY request that it be ordered and specify the semantics of the ordering by using the new Ordering-Type header (defined below) with a MKCOL request.

For collections that are ordered, the client SHOULD identify the semantics of the ordering with a URI in the Ordering-Type header, although the client MAY simply set the header value to DAV:custom to indicate that the collection is ordered but the semantics of the ordering are not being advertised. Setting the value to a URI that identifies the ordering semantics provides the information a human user or software package needs to insert new collection members into the ordering intelligently. Although the URI in the Ordering-Type header MAY point to a resource that contains a definition of the semantics of the ordering, clients SHOULD NOT access that resource to avoid overburdening its server. A value of DAV:unordered in the Ordering-Type header indicates that the client wants the collection to be unordered. If the Ordering-Type header is not present, the collection will be unordered.

Additional Marshalling:

```
Ordering-Type = "Ordering-Type" ":" absoluteURI
; absoluteURI: see RFC2396, section 3
```

The URI "DAV:unordered" indicates that the collection is not ordered, while "DAV:custom" indicates that the collection is to be ordered, but the semantics of the ordering is not being advertised. Any other URI value indicates that the collection is ordered, and identifies the semantics of the ordering.

Additional Preconditions:

(DAV:ordered-collections-supported): the server MUST support ordered collections in the part of the URL namespace identified by the request URL.

Additional Postconditions:

(DAV:ordering-type-set): if the Ordering-Type header was present, the request MUST have created a new collection resource with the DAV:ordering-type being set according to the Ordering-Type request header. The collection MUST be ordered unless the ordering type is "DAV:unordered".

5.2. Example: Creating an Ordered Collection

>> Request:

```
MKCOL /theNorth/ HTTP/1.1
Host: example.org
Ordering-Type: http://example.org/orderings/compass.html
```

>> Response:

```
HTTP/1.1 201 Created
```

In this example, a new ordered collection was created. Its DAV:ordering-type property has the URI from the Ordering-Type header as its value http://example.org/orderings/compass.html. In this case, the URI identifies the semantics governing a client-maintained ordering. As new members are added to the collection, clients or end users can use the semantics to determine where to position the new members in the ordering.

6. Setting the Position of a Collection Member

6.1. Overview

When a new member is added to a collection with a client-maintained ordering (for example, with PUT, COPY, or MKCOL), its position in the ordering can be set with the new Position header. The Position header allows the client to specify that an internal member URI should be first in the collection’s ordering, last in the collection’s ordering, immediately before some other internal member URI in the collection’s ordering, or immediately after some other internal member URI in the collection’s ordering.

If the Position request header is not used when adding a member to an ordered collection, then:

- o If the request is replacing an existing resource, the server MUST preserve the present ordering.

- o If the request is adding a new internal member URI to the collection, the server MUST append the new member to the end of the ordering.

Note to implementers: this specification does not mandate a specific implementation of MOVE operations within the same parent collection. Therefore, servers may either implement this as a simple rename operation (preserving the collection member’s position), or as a sequence of "remove" and "add" (causing the semantics of "adding a

new member" to apply). Future revisions of this specification may specify this behaviour more precisely based on future implementation experience.

Additional Marshalling:

```
Position = "Position" ":" ("first" | "last" |
                           (("before" | "after") segment))
```

segment is defined in Section 3.3 of [RFC2396].

The segment is interpreted relative to the collection to which the new member is being added.

When the Position header is present, the server MUST insert the new member into the ordering at the specified location.

The "first" keyword indicates that the new member is placed in the beginning position in the collection’s ordering, while "last" indicates that the new member is placed in the final position in the collection’s ordering. The "before" keyword indicates that the new member is added to the collection’s ordering immediately prior to the position of the member identified in the segment. Likewise, the "after" keyword indicates that the new member is added to the collection’s ordering immediately following the position of the member identified in the segment.

If the request is replacing an existing resource and the Position header is present, the server MUST remove the internal member URI from its current position, and insert it at the newly requested position.

Additional Preconditions:

(DAV:collection-must-be-ordered): the target collection MUST be ordered.

(DAV:segment-must-identify-member): the referenced segment MUST identify a resource that exists and is different from the affected resource.

Additional Postconditions:

(DAV:position-set): if a Position header is present, the request MUST create the new collection member at the specified position.

6.2. Examples: Setting the Position of a Collection Member

>> Request:

```
COPY /~user/dav/spec08.html HTTP/1.1
Host: example.org
Destination: http://example.org/~slein/dav/spec08.html
Position: after requirements.html
```

>> Response:

```
HTTP/1.1 201 Created
```

This request resulted in the creation of a new resource at example.org/~slein/dav/spec08.html. The Position header in this example caused the server to set its position in the ordering of the /~slein/dav/ collection immediately after requirements.html.

>> Request:

```
MOVE /i-d/draft-webdav-prot-08.txt HTTP/1.1
Host: example.org
Destination: http://example.org/~user/dav/draft-webdav-prot-08.txt
Position: first
```

>> Response:

```
HTTP/1.1 409 Conflict
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:error xmlns:D="DAV:">
  <D:collection-must-be-ordered/>
</D:error>
```

In this case, the server returned a 409 (Conflict) status code because the /~user/dav/ collection is an unordered collection. Consequently, the server was unable to satisfy the Position header.

6.3. Examples: Renaming a member of an ordered collection

The following sequence of requests will rename a collection member while preserving its position, independently of how the server implements the MOVE operation:

1. PROPFIND collection with depth 1, retrieving the DAV:ordering-type property (an interactive client has already likely done this in order to display the collection’s content).

2. If the DAV:ordering-type property is present and does not equal "dav:unordered" (thus if the collection is ordered), determine the current position (such as "first" or "after x") and setup the Position header accordingly.

3. Perform the MOVE operation, optionally supplying the Position header computed in the previous step.

7. Changing a Collection Ordering: ORDERPATCH method

The ORDERPATCH method is used to change the ordering semantics of a collection, to change the order of the collection’s members in the ordering, or both.

The server MUST apply the changes in the order they appear in the order XML element. The server MUST either apply all the changes or apply none of them. If any error occurs during processing, all executed changes MUST be undone and a proper error result returned.

If an ORDERPATCH request changes the ordering semantics, but does not completely specify the order of the collection members, the server MUST assign a position in the ordering to each collection member for which a position was not specified. These server-assigned positions MUST follow the last position specified by the client. The result is that all members for which the client specified a position are at the beginning of the ordering, followed by any members for which the server assigned positions. Note that the ordering of the server-assigned positions is not defined by this document, therefore servers can use whatever rule seems reasonable (for instance, alphabetically or by creation date).

If an ORDERPATCH request does not change the ordering semantics, any member positions not specified in the request MUST remain unchanged.

A request to reposition a collection member to the same place in the ordering is not an error.

If an ORDERPATCH request fails, the server state preceding the request MUST be restored.

Additional Marshalling:

The request body MUST be DAV:orderpatch element.

```
<!ELEMENT orderpatch (ordering-type?, order-member*) >

<!ELEMENT order-member (segment, position) >
<!ELEMENT position (first | last | before | after)>
<!ELEMENT segment (#PCDATA)>
<!ELEMENT first EMPTY >
<!ELEMENT last EMPTY >
<!ELEMENT before segment >
<!ELEMENT after segment >
```

PCDATA value: segment, as defined in section 3.3 of [RFC2396].

The DAV:ordering-type property is modified according to the DAV:ordering-type element.

The ordering of internal member URIs in the collection identified by the Request-URI is changed based on instructions in the order-member XML elements. Specifically, in the order that they appear in the request. The order-member XML elements identify the internal member URIs whose positions are to be changed, and describe their new positions in the ordering. Each new position can be specified as first in the ordering, last in the ordering, immediately before some other internal member URI, or immediately after some other internal member URI.

If a response body for a successful request is included, it MUST be a DAV:orderpatch-response XML element. Note that this document does not define any elements for the ORDERPATCH response body, but the DAV:orderpatch-response element is defined to ensure interoperability between future extensions that do define elements for the ORDERPATCH response body.

```
<!ELEMENT orderpatch-response ANY>
```

Since multiple changes can be requested in a single ORDERPATCH request, the server MUST return a 207 (Multi-Status) response (defined in [RFC2518]), containing DAV:response elements for either the request-URI (when the DAV:ordering-type could not be modified) or URIs of collection members to be repositioned (when an individual positioning request expressed as DAV:order-member could not be fulfilled) if any problems are encountered.

Preconditions:

(DAV:collection-must-be-ordered): see Section 6.1.

(DAV:segment-must-identify-member): see Section 6.1.

Postconditions:

(DAV:ordering-type-set): if the request body contained a DAV:ordering-type element, the request MUST have set the DAV:ordering-type property of the collection to the value specified in the request.

(DAV:ordering-modified): if the request body contained DAV:order-member elements, the request MUST have set the ordering of internal member URIs in the collection identified by the request-URI based upon the instructions in the DAV:order-member elements.

7.1. Example: Changing a Collection Ordering

Consider an ordered collection /coll-1, with bindings ordered as follows:

```
three.html
four.html
one.html
two.html
```

>> Request:

```
ORDERPATCH /coll-1/ HTTP/1.1
Host: example.org
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx

<?xml version="1.0" ?>
<d:orderpatch xmlns:d="DAV:">
   <d:ordering-type>
      <d:href>http://example.org/inorder.ord</d:href>
   </d:ordering-type>
   <d:order-member>
      <d:segment>two.html</d:segment>
      <d:position><d:first/></d:position>
   </d:order-member>
   <d:order-member>
      <d:segment>one.html</d:segment>
      <d:position><d:first/></d:position>
   </d:order-member>
```

```
      <d:order-member>
         <d:segment>three.html</d:segment>
         <d:position><d:last/></d:position>
      </d:order-member>
      <d:order-member>
         <d:segment>four.html</d:segment>
         <d:position><d:last/></d:position>
      </d:order-member>
   </d:orderpatch>

    >> Response:

   HTTP/1.1 200 OK
```

In this example, after the request has been processed, the collection's ordering semantics are identified by the URI http://example.org/inorder.ord. The value of the collection's DAV:ordering-type property has been set to this URI. The request also contains instructions for changing the positions of the collection's internal member URIs in the ordering to comply with the new ordering semantics. As the DAV:order-member elements are required to be processed in the order they appear in the request, two.html is moved to the beginning of the ordering, and then one.html is moved to the beginning of the ordering. Then three.html is moved to the end of the ordering, and finally four.html is moved to the end of the ordering. After the request has been processed, the collection's ordering is as follows:

```
one.html
two.html
three.html
four.html
```

7.2. Example: Failure of an ORDERPATCH Request

Consider a collection /coll-1/ with members ordered as follows:

```
nunavut.map
nunavut.img
baffin.map
baffin.desc
baffin.img
iqaluit.map
nunavut.desc
iqaluit.img
iqaluit.desc
```

```
 >> Request:

ORDERPATCH /coll-1/ HTTP/1.1
Host: www.nunanet.com
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx

<?xml version="1.0" ?>
<d:orderpatch xmlns:d="DAV:">
   <d:order-member>
      <d:segment>nunavut.desc</d:segment>
      <d:position>
         <d:after>
            <d:segment>nunavut.map</d:segment>
         </d:after>
      </d:position>
   </d:order-member>
   <d:order-member>
      <d:segment>iqaluit.map</d:segment>
      <d:position>
         <d:after>
            <d:segment>pangnirtung.img</d:segment>
         </d:after>
      </d:position>
   </d:order-member>
</d:orderpatch>

 >> Response:

HTTP/1.1 207 Multi-Status
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx

<?xml version="1.0" ?>
<d:multistatus xmlns:d="DAV:">
  <d:response>
    <d:href>http://www.nunanet.com/coll-1/iqaluit.map</d:href>
    <d:status>HTTP/1.1 403 Forbidden</d:status>
    <d:responsedescription>
      <d:error><d:segment-must-identify-member/></d:error>
      pangnirtung.img is not a collection member.
    </d:responsedescription>
  </d:response>
</d:multistatus>
```

In this example, the client attempted to position iqaluit.map after a URI that is not an internal member of the collection /coll-1/. The server responded to this client error with a 403 (Forbidden) status code, indicating the failed precondition DAV:segment-must-identify-member. Because ORDERPATCH is an atomic method, the request to reposition nunavut.desc (which would otherwise have succeeded) failed as well, but does not need to be expressed in the multistatus response body.

8. Listing the Members of an Ordered Collection

A PROPFIND request is used to retrieve a listing of the members of an ordered collection, just as it is used to retrieve a listing of the members of an unordered collection.

However, when responding to a PROPFIND on an ordered collection, the server MUST order the response elements according to the ordering defined on the collection. If a collection is unordered, the client cannot depend on the repeatability of the ordering of results from a PROPFIND request.

In a response to a PROPFIND with Depth: infinity, members of different collections may be interleaved. That is, the server is not required to do a breadth-first traversal. The only requirement is that the members of any ordered collection appear in the order defined for that collection. Thus, for the hierarchy illustrated in the following figure, where collection A is an ordered collection with the ordering B C D,

```
                A
               /|\
              / | \
             B  C  D
            /  /|\
           E  F G H
```

it would be acceptable for the server to return response elements in the order A B E C F G H D or "A B E C H G F D" as well (if C is unordered). In this response, B, C, and D appear in the correct order, separated by members of other collections. Clients can use a series of Depth: 1 PROPFIND requests to avoid the complexity of processing Depth: infinity responses based on depth-first traversals.

8.1. Example: PROPFIND on an Ordered Collection

Suppose a PROPFIND request is submitted to /MyColl/, which has its members ordered as follows.

```
/MyColl/
   lakehazen.html
   siorapaluk.html
   iqaluit.html
   newyork.html
```

>> Request:

```
PROPFIND /MyColl/ HTTP/1.1

Host: example.org
Depth: 1
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" ?>
<D:propfind xmlns:D="DAV:">
  <D:prop xmlns:J="http://example.org/jsprops/">
    <D:ordering-type/>
    <D:resourcetype/>
    <J:latitude/>
 </D:prop>
</D:propfind>
```

>> Response:

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

<?xml version="1.0" ?>
<D:multistatus xmlns:D="DAV:"
               xmlns:J="http://example.org/jsprops/">
   <D:response>
      <D:href>http://example.org/MyColl/</D:href>
      <D:propstat>
         <D:prop>
            <D:ordering-type>
               <D:href>DAV:custom</D:href>
            </D:ordering-type>
            <D:resourcetype><D:collection/></D:resourcetype>
         </D:prop>
         <D:status>HTTP/1.1 200 OK</D:status>
```

```
         </D:propstat>
         <D:propstat>
            <D:prop>
               <J:latitude/>
            </D:prop>
            <D:status>HTTP/1.1 404 Not Found</D:status>
         </D:propstat>
      </D:response>
      <D:response>
         <D:href>http://example.org/MyColl/lakehazen.html</D:href>
         <D:propstat>
            <D:prop>
               <D:resourcetype/>
               <J:latitude>82N</J:latitude>
            </D:prop>
            <D:status>HTTP/1.1 200 OK</D:status>
         </D:propstat>
         <D:propstat>
            <D:prop>
               <D:ordering-type/>
            </D:prop>
            <D:status>HTTP/1.1 404 Not Found</D:status>
         </D:propstat>
      </D:response>
      <D:response>
         <D:href
         >http://example.org/MyColl/siorapaluk.html</D:href>
         <D:propstat>
            <D:prop>
               <D:resourcetype/>
               <J:latitude>78N</J:latitude>
            </D:prop>
            <D:status>HTTP/1.1 200 OK</D:status>
         </D:propstat>
         <D:propstat>
            <D:prop>
               <D:ordering-type/>
            </D:prop>
            <D:status>HTTP/1.1 404 Not Found</D:status>
         </D:propstat>
      </D:response>
      <D:response>
         <D:href>http://example.org/MyColl/iqaluit.html</D:href>
         <D:propstat>
            <D:prop>
               <D:resourcetype/>
               <J:latitude>62N</J:latitude>
            </D:prop>
```

```
            <D:status>HTTP/1.1 200 OK</D:status>
         </D:propstat>
         <D:propstat>
            <D:prop>
               <D:ordering-type/>
            </D:prop>
            <D:status>HTTP/1.1 404 Not Found</D:status>
         </D:propstat>
      </D:response>
      <D:response>
         <D:href>http://example.org/MyColl/newyork.html</D:href>
         <D:propstat>
            <D:prop>
               <D:resourcetype/>
               <J:latitude>45N</J:latitude>
            </D:prop>
            <D:status>HTTP/1.1 200 OK</D:status>
         <D:propstat>
            <D:prop>
               <D:ordering-type/>
            </D:prop>
            <D:status>HTTP/1.1 404 Not Found</D:status>
         </D:propstat>
         </D:propstat>
      </D:response>
   </D:multistatus>
```

In this example, the server responded with a list of the collection members in the order defined for the collection.

9. Relationship to versioned collections

The Versioning Extensions to WebDAV [RFC3253] introduce the concept of versioned collections, recording both the dead properties and the set of internal version-controlled bindings. This section defines how this feature interacts with ordered collections.

This specification considers both the ordering type (DAV:ordering-type property) and the ordering of collection members to be part of the state of a collection. Therefore, both MUST be recorded upon CHECKIN or VERSION-CONTROL, and both MUST be restored upon CHECKOUT, UNCHECKOUT or UPDATE (where for compatibility with RFC 3253, only the ordering of version-controlled members needs to be maintained).

9.1. Collection Version Properties

9.1.1. Additional semantics for DAV:version-controlled-binding-set (protected)

For ordered collections, the DAV:version-controlled-binding elements MUST appear in the ordering defined for the checked-in ordered collection.

9.1.2. DAV:ordering-type (protected)

The DAV:ordering-type property records the DAV:ordering-type property of the checked-in ordered collection.

9.2. Additional CHECKIN semantics

Additional Postconditions:

(DAV:initialize-version-controlled-bindings-ordered): If the request-URL identified a both ordered and version-controlled collection, then the child elements of DAV:version-controlled-binding-set of the new collection version MUST appear in the ordering defined for that collection.

(DAV:initialize-collection-version-ordering-type): If the request-URL identified a both ordered and version-controlled collection, then the DAV:ordering-type property of the new collection version MUST be a copy of the collection’s DAV:ordering-type property.

9.3. Additional CHECKOUT Semantics

Additional Postconditions:

(DAV:initialize-version-history-bindings-ordered): If the request has been applied to a collection version with a DAV:ordering-type other than "DAV:unordered", the bindings in the new working collection MUST be ordered according to the collection version’s DAV:version-controlled-binding-set property.

(DAV:initialize-ordering-type): If the request has been applied to a collection version, the DAV:ordering-type property of the new working collection MUST be initialized from the collection version’s DAV:ordering-type property.

9.4. Additional UNCHECKOUT, UPDATE, and MERGE Semantics

Additional Postconditions:

(DAV:update-version-controlled-collection-members-ordered): If the request modified the DAV:checked-in version of a version-controlled collection and the DAV:ordering-type for the checked-in version is not unordered ("DAV:unordered"), the version-controlled members MUST be ordered according to the checked-in version's DAV:version-controlled-binding-set property. The ordering of non-version-controlled members is server-defined.

(DAV:update-version-ordering-type): If the request modified the DAV:checked-in version of a version-controlled collection, the DAV:ordering-type property MUST be updated from the checked-in version's property.

10. Capability Discovery

Sections 9.1 and 15 of [RFC2518] describe the use of compliance classes with the DAV header in responses to OPTIONS, indicating which parts of the Web Distributed Authoring protocols the resource supports. This specification defines an OPTIONAL extension to [RFC2518]. It defines a new compliance class, called ordered-collections, for use with the DAV header in responses to OPTIONS requests. If a collection resource does support ordering, its response to an OPTIONS request may indicate that it does, by listing the new ORDERPATCH method as one it supports, and by listing the new ordered-collections compliance class in the DAV header.

When responding to an OPTIONS request, only a collection or a null resource can include ordered-collections in the value of the DAV header. By including ordered-collections, the resource indicates that its internal member URIs can be ordered. It implies nothing about whether any collections identified by its internal member URIs can be ordered.

Furthermore, RFC 3253 [RFC3253] introduces the live properties DAV:supported-method-set (section 3.1.3) and DAV:supported-live-property-set (section 3.1.4). Servers MUST support these properties as defined in RFC 3253.

10.1. Example: Using OPTIONS for the Discovery of Support for Ordering

```
 >> Request:

OPTIONS /somecollection/ HTTP/1.1
Host: example.org

 >> Response:

HTTP/1.1 200 OK
Allow: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, COPY, MOVE
Allow: MKCOL, PROPFIND, PROPPATCH, LOCK, UNLOCK, ORDERPATCH
DAV: 1, 2, ordered-collections
```

The DAV header in the response indicates that the resource /somecollection/ is level 1 and level 2 compliant, as defined in [RFC2518]. In addition, /somecollection/ supports ordering. The Allow header indicates that ORDERPATCH requests can be submitted to /somecollection/.

10.2. Example: Using Live Properties for the Discovery of Ordering

```
 >> Request:
PROPFIND /somecollection HTTP/1.1
Depth: 0
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx

<?xml version="1.0" encoding="UTF-8" ?>
<propfind xmlns="DAV:">
  <prop>
    <supported-live-property-set/>
    <supported-method-set/>
  </prop>
</propfind>

 >> Response:
HTTP/1.1 207 Multi-Status
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx

<?xml version="1.0" encoding="utf-8" ?>
<multistatus xmlns="DAV:">
  <response>
    <href>http://example.org/somecollection</href>
    <propstat>
      <prop>
```

```
          <supported-live-property-set>
            <supported-live-property>
              <prop><ordering-type/></prop>
            </supported-live-property>
            <!-- ... other live properties omitted for brevity ... -->
          </supported-live-property-set>
          <supported-method-set>
            <supported-method name="COPY" />
            <supported-method name="DELETE" />
            <supported-method name="GET" />
            <supported-method name="HEAD" />
            <supported-method name="LOCK" />
            <supported-method name="MKCOL" />
            <supported-method name="MOVE" />
            <supported-method name="OPTIONS" />
            <supported-method name="ORDERPATCH" />
            <supported-method name="POST" />
            <supported-method name="PROPFIND" />
            <supported-method name="PROPPATCH" />
            <supported-method name="PUT" />
            <supported-method name="TRACE" />
            <supported-method name="UNLOCK" />
          </supported-method-set>
        </prop>
        <status>HTTP/1.1 200 OK</status>
      </propstat>
    </response>
  </multistatus>
```

Note that actual responses MUST contain a complete list of supported live properties.

11. Security Considerations

This section is provided to make WebDAV implementers aware of the security implications of this protocol.

All of the security considerations of HTTP/1.1 and the WebDAV Distributed Authoring Protocol specification also apply to this protocol specification. In addition, ordered collections introduce a new security concern. This issue is detailed here.

11.1. Denial of Service and DAV:ordering-type

There may be some risk of denial of service at sites that are advertised in the DAV:ordering-type property of collections. However, it is anticipated that widely-deployed applications will use hard-coded values for frequently-used ordering semantics rather than

looking up the semantics at the location specified by DAV:ordering-type. This risk will be further reduced if clients observe the recommendation of Section 5.1 that requests not be sent to the URI in DAV:ordering-type.

12. Internationalization Considerations

This specification follows the practices of [RFC2518] by encoding all human-readable content using [XML] and in the treatment of names. Consequently, this specification complies with the IETF Character Set Policy [RFC2277].

WebDAV applications MUST support the character set tagging, character set encoding, and the language tagging functionality of the XML specification. This constraint ensures that the human-readable content of this specification complies with [RFC2277].

As in [RFC2518], names in this specification fall into three categories: names of protocol elements such as methods and headers, names of XML elements, and names of properties. The naming of protocol elements follows the precedent of HTTP using English names encoded in USASCII for methods and headers. The names of XML elements used in this specification are English names encoded in UTF-8.

For error reporting, [RFC2518] follows the convention of HTTP/1.1 status codes, including with each status code a short, English description of the code (e.g., 423 Locked). Internationalized applications will ignore this message, and display an appropriate message in the user’s language and character set.

This specification introduces no new strings that are displayed to users as part of normal, error-free operation of the protocol.

For the rationale of these decisions and advice for application implementers, see [RFC2518].

13. IANA Considerations

This document uses the namespaces defined by [RFC2518] for properties and XML elements. All other IANA considerations mentioned in [RFC2518] also apply to this document.

14. Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights which may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

15. Contributors

This document has benefited from significant contributions from Geoff Clemm, Jason Crawford, Jim Davis, Chuck Fay and Judith Slein.

16. Acknowledgements

This document has benefited from thoughtful discussion by Jim Amsden, Steve Carter, Tyson Chihaya, Ken Coar, Ellis Cohen, Bruce Cragun, Spencer Dawkins, Mark Day, Rajiv Dulepet, David Durand, Lisa Dusseault, Roy Fielding, Yaron Goland, Fred Hitt, Alex Hopmann, Marcus Jager, Chris Kaler, Manoj Kasichainula, Rohit Khare, Daniel LaLiberte, Steve Martin, Larry Masinter, Jeff McAffer, Surendra Koduru Reddy, Max Rible, Sam Ruby, Bradley Sergeant, Nick Shelness, John Stracke, John Tigue, John Turner, Kevin Wiggen, and others.

17. Normative References

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

[RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[RFC2396] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[RFC2518] Goland, Y., Whitehead, E., Faizi, A., Carter, S. and D. Jensen, "HTTP Extensions for Distributed Authoring -- WEBDAV", RFC 2518, February 1999.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3253] Clemm, G., Amsden, J., Ellison, T., Kaler, C. and J. Whitehead, "Versioning Extensions to WebDAV (Web Distributed Authoring and Versioning)", RFC 3253, March 2002.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C. and E. Maler, "Extensible Markup Language (XML) 1.0 (2nd ed)", W3C REC-xml, October 2000, <http://www.w3.org/TR/2000/REC-xml-20001006>.

Appendix A. Extensions to the WebDAV Document Type Definition

```
<!ELEMENT orderpatch (ordering-type?, order-member*) >
<!ELEMENT order-member (segment, position) >
<!ELEMENT ordering-type (href) >
<!ELEMENT position (first | last | before | after)>
<!ELEMENT first EMPTY >
<!ELEMENT last EMPTY >
<!ELEMENT before segment >
<!ELEMENT after segment >
<!ELEMENT segment (#PCDATA)>
```

Index

C
- Client-Maintained Ordering 4
- Condition Names
  - DAV:collection-must-be-ordered (pre) 9
  - DAV:initialize-collection-version-ordering-type (post) 20
  - DAV:initialize-ordering-type (post) 21
  - DAV:initialize-version-controlled-bindings-ordered (post) 20
  - DAV:initialize-version-history-bindings-ordered (post) 20
  - DAV:ordered-collections-supported (pre) 7
  - DAV:ordering-modified (post) 13
  - DAV:ordering-type-set (post) 7, 13
  - DAV:position-set (post) 9
  - DAV:segment-must-identify-member (pre) 9
  - DAV:update-version-controlled-collection-members-ordered (post) 21
  - DAV:update-version-ordering-type (post) 21

D
- DAV header
  - compliance class ’ordered-collections’ 21
- DAV:collection-must-be-ordered precondition 9
- DAV:custom ordering type 6
- DAV:initialize-collection-version-ordering-type postcondition 20
- DAV:initialize-ordering-type postcondition 21
- DAV:initialize-version-controlled-bindings-ordered postcondition 20
- DAV:initialize-version-history-bindings-ordered postcondition 20
- DAV:ordered-collections-supported precondition 7
- DAV:ordering-modified postcondition 13
- DAV:ordering-type property 6
- DAV:ordering-type-set postcondition 7, 13
- DAV:position-set postcondition 9
- DAV:segment-must-identify-member precondition 9
- DAV:unordered ordering type 6

DAV:update-version-controlled-collection-members-ordered
postcondition 21
DAV:update-version-ordering-type postcondition 21

H
Headers
Ordering-Type 7
Position 9

M
Methods
ORDERPATCH 11

O
Ordered Collection 4
Ordering Semantics 5
Ordering-Type header 7
ORDERPATCH method 11

P
Position header 9
Properties
DAV:ordering-type 6

S
Server-Maintained Ordering 5

U
Unordered Collection 4

Authors' Addresses

Jim Whitehead
UC Santa Cruz, Dept. of Computer Science
1156 High Street
Santa Cruz, CA 95064
US

EMail: ejw@cse.ucsc.edu

Julian F. Reschke, Ed.
greenbytes GmbH
Salzmannstrasse 152
Muenster, NW 48159
Germany

Phone: +49 251 2807760
Fax: +49 251 2807761
EMail: julian.reschke@greenbytes.de
URI: http://greenbytes.de/tech/webdav/

Full Copyright Statement

Copyright (C) The Internet Society (2003). All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assignees.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.