WEBDAV Working Group   J. Slein, Xerox
INTERNET DRAFT   E.J. Whitehead Jr., UC Irvine
<draft-ietf-webdav-ordering-protocol-02.txt>   J. Davis, CourseNet
                                                         G. Clemm, Rational
                                                            C. Fay, FileNet
                                                           J. Crawford, IBM
                                                          December 20, 1999
Expires June 20, 2000

WebDAV Ordered Collections Protocol

Status of this Memo

Abstract

This specification extends the WebDAV Distributed Authoring Protocol to support 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

Since this document describes a set of extensions to the WebDAV Distributed Authoring Protocol [WebDAV], 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]. 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].

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 where 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, which 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 ordering. The protocol is designed to allow support to be added 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. Sections 9 through 11 define the headers, properties, and XML elements needed to support ordered collections. Section 12 describes capability discovery. Sections 13 through 15 discuss security, internationalization, and IANA considerations. The remaining sections provide supporting information.

3 Terminology

The terminology used here follows that in the WebDAV Distributed Authoring Protocol specification [WebDAV]. Definitions of the terms resource, Uniform Resource Identifier (URI), and Uniform Resource Locator (URL) are provided in [URI].

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

4 Overview of Ordered Collections

If a collection is unordered, 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) or later (using the ORDERPATCH 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 future extension to server-maintained orderings.

A collection that supports ordering is not required to be ordered. It is up to the client to decide whether a given collection is ordered and, if so, to specify the semantics to be used for ordering its members.

If a collection is ordered, each of its internal member URIs MUST be in the ordering exactly once, and the ordering MUST NOT include any URI that is not an internal member 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. The server MUST 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.

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 Ordered header (defined in Section 9.1) with a MKCOL request.

For collections that are ordered, the client SHOULD identify the semantics of the ordering with a URI in the Ordered 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 semanticsprovides the information a human user or software package needs to insert new collection members into the ordering intelligently. Although the URI in the Ordered header MAY point to a resource that contains a definition of the semantics of the ordering, clients SHOULD NOT access that resource, in order to avoid overburdening its server. A value of DAV:unordered in the Ordering header indicates that the client wants the collection to be unordered. If the Ordered header is not present, the collection will be unordered.
Every collection that supports ordering MUST have a DAV:orderingtype property (defined in Section 10.1), which indicates whether the collection is ordered and, if so, identifies the semantics of the ordering. The server sets the initial value of this property based on the value of the Ordering header in the MKCOL request, if any. If the

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

5.2 Example: Creating an Ordered Collection

```
>> Request:

MKCOL /theNorth/ HTTP/1.1
Host: www.server.org
Ordered: <http://www.server.org/orderings/compass.html>

>> Response:

HTTP/1.1 201 Created
```

In this example a new, ordered collection was created. Its DAV:orderingtype property has as its value the URI from the Ordered header, http://www.server.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, MKREF, COPY, or MKCOL), its position in the ordering can be set with the new Position header (defined in Section 9.2). 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.

6.2 Status Codes

409 (Conflict): Several conditions may cause this response. The request may specify a position that is before or after a URI that is not an internal member URI of the collection, or before or after itself. The request may attempt to specify the new member's position in an unordered collection.

6.3 Examples: Setting the Position of a Collection Member

```
>> Request:

COPY /~whitehead/dav/spec08.html HTTP/1.1
Host: www.ics.uci.edu
Destination: http://www.xerox.com/~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 www.xerox.com/~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-protocol-08.txt HTTP/1.1
Host: www.ics.uci.edu
Destination: http://www.ics.uci.edu/~whitehead/dav/draft-webdav-
     protocol-08.txt
Position: first

>> Response:

HTTP/1.1 409 Conflict
```

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

7 Changing a Collection Ordering

7.1 ORDERPATCH Method

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

The ORDERPATCH method changes the ordering semantics of the collection identified by the Request-URI, based on the value of DAV:orderingtype submitted in the request entity body.

The ORDERPATCH method alters the ordering of internal member URIs in the collection identified by the Request-URI, based on instructions in the ordermember XML elements in the request entity body. The ordermember 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.

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 all follow the last one 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.

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

7.1.1 Status Codes

Since multiple changes can be requested in a single ORDERPATCH request, if any problems are encountered, the server MUST return a 207 (Multi-Status) response, as defined in [WebDAV].

The following are examples of response codes one would expect to be used in a 207 (Multi-Status) response for this method:

200 (OK): The change in ordering was successfully made.

409 (Conflict): Several conditions may cause this response. The request may specify a position that is before or after a URI that is not an internal member URI of the collection, or before or after itself. The request may attempt to set the positions of members of an unordered collection.

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

7.1.2 Example: Changing a Collection Ordering

Consider a collection /coll-1/ whose DAV:orderingtype is DAV:whim, with bindings ordered as follows:

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

>> Request:

```
ORDERPATCH /coll-1/ HTTP/1.1
Host: www.myserver.com
Content-Type: text/xml
Content-Length: xxx

<?xml version="1.0" ?>
<d:order xmlns:d="DAV:">
   <d:orderingtype>
      <d:href>http://www.myserver.com/inorder.ord</d:href>
   </d:orderingtype>
   <d:ordermember>
      <d:href>two.html</d:href>
      <d:position>
         <d:first/>
      </d:position>
   </d:ordermember>
   <d:ordermember>
      <d:href>one.html</d:href>
      <d:position>
         <d:first/>
      </d:position>
   </d:ordermember>
   <d:ordermember>
      <d:href>three.html</d:href>
      <d:position>
         <d:last/>
      </d:position>
   </d:ordermember>
   <d:ordermember>
      <d:href>four.html</d:href>
      <d:position>
         <d:last/>
      </d:position>
   </d:ordermember>
</d:order>
```

```
>> 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://www.myserver.com/inorder.ord. The value of the collection's DAV:orderingtype 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. If href elements are relative URIs, as in this example, they are interpreted relative to the collection whose ordering is being modified. The DAV:ordermember elements are required to be processed in the order they appear in the request. Consequently, 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
```

7.1.3 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
Content-Length: xxx

<?xml version="1.0" ?>
```

```
<d:order xmlns:d="DAV:">
   <d:ordermember>
      <d:href>nunavut.desc</d:href>
      <d:position>
         <d:after>
            <d:segment>nunavut.map</d:segment>
         </d:after>
      </d:position>
   </d:ordermember>
   <d:ordermember>
      <d:href>iqaluit.map</d:href>
      <d:position>
         <d:after>
            <d:segment>pangnirtung.img</d:segment>
         </d:after>
      </d:position>
```

```
   </d:ordermember>
</d:order>
```

>> Response:

```
HTTP/1.1 207 Multi-Status
Content-Type: text/xml
Content-Length: xxx

<?xml version="1.0" ?>
<d:multistatus xmlns:d="DAV:">
   <d:response>
      <d:href>http://www.nunanet.com/coll-1/nunavut.desc</d:href>
      <d:status>HTTP/1.1 424 Failed Dependency</d:status>
   </d:response>
   <d:response>
      <d:href>http://www.nunanet.com/coll-1/iqaluit.map</d:href>
      <d:status>HTTP/1.1 409 Conflict</d:status>
      <d:responsedescription>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 409 (Conflict) status code. Because ORDERPATCH is an atomic method, the request to reposition nunavut.desc (which would otherwise have succeeded) failed with a 424 (Failed Dependency) status code.

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 the 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. 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 /MyCollection/, which has its members ordered as follows.

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

>> Request:

```
PROPFIND /MyCollection/ HTTP/1.1
Host: www.svr.com
Depth: 1
```

```
Content-Type: text/xml
Content-Length: xxxx

<?xml version="1.0" ?>
<D:propfind xmlns:D="DAV:">
   <D:prop xmlns:J=http://www.svr.com/jsprops/>
      <D:resourcetype/>
      <J:latitude/>
   </D:prop>
</D:propfind>

>> Response:

HTTP/1.1 207 Multi-Status
Content-Type: text/xml
Content-Length: xxxx

<?xml version="1.0" ?>
<D:multistatus xmlns:D="DAV:"
               xmlns:J="http:www.svr.com/jsprops/">
   <D:response>
      <D:href>http://www.svr.com/MyCollection/</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>
            <J:latitude/>
```

```
         </D:prop>
         <D:status>HTTP/1.1 404 Not Found</D:status>
      </D:propstat>
   </D:response>
   <D:response>
      <D:href>http://www.svr.com/MyCollection/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:response>
   <D:response>
      <D:href>http://www.svr.com/MyCollection/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:response>
   <D:response>
      <D:href>http://www.svr.com/MyCollection/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:response>
   <D:response>
      <D:href>http://www.svr.com/MyCollection/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:response>
</D:multistatus>
```

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

9 Headers

9.1 Ordered Entity Header

```
Ordered = "Ordered" ":" ("DAV:unordered" | "DAV:custom" | Coded-url)
```

The Ordered header may be used with MKCOL to request that the new collection be ordered and to specify its ordering semantics. A value of "DAV:unordered" indicates that the collection is not ordered. A value of "DAV:custom" indicates that the collection is to be ordered, but the semantics of the ordering is not being advertised. Any other Coded-url value indicates that the collection is ordered, and identifies the semantics of the ordering.

9.2 Position Request Header

```
Position = "Position" ":" ("first" | "last" |
                           (("before" | "after") segment))
segment is defined in Section 3.3 of [URI].
```

The Position header may be used with any method that adds a member to an ordered collection, to tell the server where in the collection ordering to position the new member being added to the collection. Examples of methods that add members to collections are BIND, PUT, COPY, MOVE, etc.

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

The server MUST insert the new member into the ordering at the location specified in the Position header, if one is present (and if the collection is ordered).

The "first" keyword indicates the new member is put in the beginning position in the collection's ordering, while "last" indicates the new member is put in the final position in the collection's ordering. The "before" keyword indicates 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 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 previous position, and then insert it at the requested position.

If an attempt is made to use the Position header on a collection that is unordered, the server MUST fail the request with a 409 (Conflict) status code.

10 Properties

10.1 orderingtype Property

Name:	orderingtype
Namespace:	DAV:
Purpose:	Indicates whether the collection is ordered and, if so, uniquely identifies the semantics of the ordering being used. 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 Ordered header with a MKCOL request
            or by submitting an ORDERPATCH request.
Value:      The value unordered indicates that the collection is not
            ordered. The value custom indicates that the collection is
            ordered, but the semantics governing the ordering are not
            being advertised.  If the value is an href element, it
            contains a URI that uniquely identifies the semantics of the
            collection's ordering.
```

```
<!ELEMENT orderingtype (unordered | custom | href) >
```

11 XML Elements

11.1 unordered XML Element

```
Name:       unordered
Namespace:  DAV:
Purpose:    A value of the DAV:orderingtype property that 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.
```

```
<!ELEMENT unordered EMPTY >
```

11.2 custom XML Element

```
Name:       custom
Namespace:  DAV:
Purpose:    A value of the DAV:orderingtype property that indicates that
            the collection is ordered, but the semantics of the ordering
            are not being advertised.
```

```
<!ELEMENT custom EMPTY >
```

11.3 order XML Element

```
Name:       order
Namespace:  DAV:
Purpose:    For use with the new ORDERPATCH method.  Describes a change
            to be made in a collection's ordering semantics or in the
            positions of its members in the ordering or both.
Value:      An optional identifier of an ordering semantics for the
            collection, followed by a list of changes to be made in the
            positions of the members in the collection's ordering.
```

```
<!ELEMENT order (orderingtype?, ordermember*) >
```

11.4 ordermember XML Element

```
Name:       ordermember
Namespace:  DAV:
Purpose:    Occurs in the order XML element, and describes the new
```

position of a single internal member URI in the collection's ordering.

Value: An href containing a member's path segment, and a description of its new position in the ordering. The href XML element is defined in [WebDAV], Section 11.3.

```
<!ELEMENT ordermember (href, position) >
```

11.5 position XML Element

Name:	position
Namespace:	DAV:
Purpose:	Occurs in the ordermember XML element. Describes the new position in a collection's ordering of one of the members it contains.
Value:	The new position can be described as first in the collection's ordering, last in the collection's ordering, immediately before some other collection member, or immediately after some other collection member.

```
<!ELEMENT position (first | last | before | after)>
```

11.6 first XML Element

Name:	first
Namespace:	DAV:
Purpose:	Occurs in the position XML element. Specifies that the member should be placed first in the collection's ordering.

```
<!ELEMENT first EMPTY >
```

11.7 last XML Element

Name:	last
Namespace:	DAV:
Purpose:	Occurs in the position XML element. Specifies that the member should be placed last in the collection's ordering.

```
<!ELEMENT last EMPTY >
```

11.8 before XML Element

Name:	before
Namespace:	DAV:
Purpose:	Occurs in the position XML element. Specifies that the member should be placed immediately before the member in the enclosed segment XML element in the collection's ordering.
Value:	URI (relative to the parent collection) of the member

```
             it precedes in the ordering

<!ELEMENT before segment >
```

11.9 after XML Element

```
Name:        after
Namespace:   DAV:
Purpose:     Occurs in the position XML element.  Specifies that the
```

```
             member should be placed immediately after the member in the
             enclosed segment XML element in the collection's ordering.
Value:       URI (relative to the parent collection) of the member
             it follows in the ordering

<!ELEMENT after segment >
```

11.10 segment XML Element

```
Name:        segment
Namespace:   DAV:
Purpose:     Identifies a member of a collection, used in the DAV:before
             and DAV:after elements, to define one member's position in
             a collection ordering relative to another member of the
             collection.
Value:       segment   ; as defined in section 3.3 of [URI].

<!ELEMENT segment (#PCDATA)>
```

12 Capability Discovery

Sections 9.1 and 15 of [WebDAV] describe the use of compliance classes with the DAV header in responses to OPTIONS, to indicate which parts of the Web Distributed Authoring protocols the resource supports. This specification defines an OPTIONAL extension to [WebDAV]. It defines a new compliance class, called orderedcoll, 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 orderedcoll compliance class in the DAV header.

When responding to an OPTIONS request, only a collection or a null resource can include orderedcoll in the value of the DAV header. By including orderedcoll, 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.

12.1 Example: Discovery of Support for Ordering

```
>> Request:

OPTIONS /somecollection/ HTTP/1.1
HOST: somehost.org

>> Response:

HTTP/1.1 200 OK
Date: Tue, 20 Jan 1998 20:52:29 GMT
Connection: close
Accept-Ranges: none
Allow: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, COPY, MOVE, MKCOL,
PROPFIND, PROPPATCH, LOCK, UNLOCK, ORDERPATCH
Public: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, COPY, MOVE, MKCOL,
PROPFIND, PROPPATCH, LOCK, UNLOCK, BIND, MKREF, ORDERPATCH
DAV: 1, 2, orderedcoll
```

The DAV header in the response indicates that the resource /somecollection/ is level 1 and level 2 compliant, as defined in [WebDAV]. In addition, /somecollection/ supports ordering. The Allow header indicates that ORDERPATCH requests can be submitted to /somecollection/. The Public header shows that other Request-URIs on the server support additional methods.

13 Security Considerations

This section is provided to make WebDAV applications 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.

13.1 Denial of Service and DAV:orderingtype

There may be some risk of denial of service at sites that are advertised in the DAV:orderingtype 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:orderingtype. This risk will be further reduced if clients observe the recommendation of Section 5.1 that they not send requests to the URI in DAV:orderingtype.

14 Internationalization Considerations

This specification follows the practices of [WebDAV] in encoding all human-readable content using XML [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 [WebDAV}, 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. 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, [WebDAV] 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 rationales for these decisions and advice for application implementors, see [WebDAV].

15 IANA Considerations

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

16 Copyright

To be supplied by the RFC Editor.

17 Intellectual Property

To be supplied by the RFC Editor.

18 Acknowledgements

This draft 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, Roy Fielding, Yaron Goland, Fred Hitt, Alex Hopmann, Marcus Jager, Chris Kaler, Manoj Kasichainula, Rohit Khare, Daniel LaLiberte, Lisa Lippert, 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.

19 References

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

[URI] T. Berners-Lee, R. Fielding, L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax." RFC 2396. MIT/LCS, U.C. Irvine, Xerox. August, 1998.

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

[XML] T. Bray, J. Paoli, C.M. Sperberg-McQueen, "Extensible Markup Language (XML)." World Wide Web Consortium Recommendation REC-xml-**19980210**. **http://www.w3.org/TR/1998/REC-xml-19980210**.

[HTTP] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1." RFC **2616**. **UC Irvine, Compaq, W3C, Xerox, Microsoft.** June, 1999.

[WebDAV] Y. Y. Goland, E. J. Whitehead, Jr., A. Faizi, S. R. Carter, D. Jensen, "HTTP Extensions for Distributed Authoring - WebDAV." RFC 2518. Microsoft, U.C. Irvine, Netscape, Novell. February, 1999.

20 Authors' Addresses

J. Slein
Xerox Corporation
800 Phillips Road, 105-50C
Webster, NY 14580
Email: jslein@crt.xerox.com

E. J. Whitehead, Jr.
Dept. of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425
Email: ejw@ics.uci.edu

J. Davis
CourseNet Systems
170 Capp Street
San Francisco, CA 94110
Email: jrd3@alum.mit.edu

G. Clemm
Rational Software Corporation
20 Maguire Road
Lexington, MA 02173-3104
Email: gclemm@rational.com

C. Fay
FileNet Corporation
3565 Harbor Boulevard
Costa Mesa, CA 92626-1420
Email: cfay@filenet.com

J. Crawford
IBM Research
P.O. Box 704
Yorktown Heights, NY 10598
Email: ccjason@us.ibm.com

21 Appendices

21.1 Appendix 1: Extensions to the WebDAV Document Type Definition

```
<!--============= XML Elements from Section 11 ================-->
<!ELEMENT unordered EMPTY >
<!ELEMENT custom EMPTY >
<!ELEMENT order (orderingtype?, ordermember*) >
<!ELEMENT ordermember (href, position) >
<!ELEMENT position (first | last | before | after)>
<!ELEMENT first EMPTY >
<!ELEMENT last EMPTY >
<!ELEMENT before segment >
<!ELEMENT after segment >
<!ELEMENT segment (#PCDATA)>
<!--============= Property Elements from Section 10 =============-->
<!ELEMENT orderingtype (unordered | custom | href) >
```