WEBDAV Working Group
INTERNET DRAFT

J. Slein, Xerox
E.J. Whitehead Jr., UC Irvine
J. Davis, CourseNet
G. Clemm, Rational
C. Fay, FileNet
J. Crawford, IBM
T. Chihaya, DataChannel

June 18, 1999

Expires December 18, 1999

WebDAV Advanced Collections Protocol
draft-ietf-webdav-collection-protocol-04.txt

Status of this Memo

Abstract

The WebDAV Distributed Authoring Protocol provides basic support for collections, offering the ability to create and list unordered collections. Many applications, however, need more powerful collections, especially for resource sharing and collection ordering.

This specification defines HTTP methods, headers, and XML elements that supplement the WebDAV Distributed Authoring Protocol to support resource sharing and collection orderings. Resource sharing is provided by bindings and redirect references. Bindings create new mappings of URIs to resources, while redirect references respond to most requests with an

HTTP Redirection (i.e., a 302 status code). An ordered collection always returns a listing of its members in a specific order. Together, these capabilities are referred to as WebDAV Advanced Collections.

Table of Contents

1 Notational Conventions

Since this document describes a set of extensions 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 Terminology

The terminology used here follows and extends 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].

Association
: A direct or indirect connection between a resource and a namespace element that supports resource sharing. The bindings, URI mappings, and redirect references defined in this specification are types of associations.

URI Mapping
: An association between an absolute URL or URI and a resource. Since a resource can represent items that are not network retrievable, as well as those that are, it is possible for a resource to have zero, one, or many URI mappings to URLs or URIs. Mapping a resource to an "http" scheme URL makes it possible to submit HTTP protocol requests to the resource using the URL.

Path Segment
: Informally, the characters found between slashes ("/") in a URL or URI. Formally, as defined in section 3.3 of [URI].

Binding
: An association between a single path segment (in a collection) and a resource. A binding creates one or more URI mappings, and hence is a mechanism for resource sharing, allowing a single resource to be accessed from multiple locations in a URI namespace.

Collection
: A resource that contains, as part of its state, a set of bindings which identify member resources.

Internal Member URI
: The URI mapping, created by a binding that is contained in a collection. While, in general, bindings can create multiple URI mappings to a resource, for a given request, only one of these URI mappings is referred to as the internal member. The URI of the parent collection used in a given request determines the base URI for internal member URI calculation.

  In [WebDAV], a collection is defined as containing a list of internal member URIs, where an internal member URI is the URI of the collection, plus a single path segment. This definition combines the two concepts of binding and mapping that are separated in this specification. As a result, this specification redefines a collection's state to be a set of bindings, and redefines an internal member URI to be a mapping derived from a binding. After this redefinition, an internal member URI can be

  used when reading [WebDAV] without loss of meaning. For purposes of interpretation, when [WebDAV] discusses a collection "containing" an internal member URI, this should be read as the collection containing a binding whose mapping to a URI creates an internal member URI, in this sense "containing" the internal member URI. The authors of this specification anticipate and recommend that future revisions of [WebDAV] perform a full reconciliation of terms between these two specifications.

Reference
: A resource whose purpose is to provide access to another resource.

Redirect Reference
: A resource whose purpose is to provide access to another resource, and that requires client action before it can be resolved. The client is aware that this type of reference is mediating between

it and the target resource.

Ordinary Resource
: A resource that is not a reference to another resource.

Target Resource
: The resource referenced by a referential resource.

Referential Integrity
: The integrity of a reference is preserved as long as it points to the same resource it pointed to when it was created. Its integrity may be destroyed if the target resource is moved without updating the reference to reflect its new location, or if the target resource is deleted while a reference to it still exists.

3 Introduction

The simple collections that the WebDAV Distributed Authoring Protocol specification supports are powerful enough to be widely useful. They provide for the hierarchical organization of resources, with mechanisms for creating and deleting collections, copying and moving them, locking them, adding members to them and removing members from them, and getting listings of their members. Delete, copy, move, list, and lock operations can be applied recursively, so that a client can operate on whole hierarchies with a single request.

Many applications, however, need more powerful collections. There are two areas in particular where more powerful functionality is often needed: shared resources and ordering. This specification defines extensions to [WebDAV] in both these areas.

Organizing resources into hierarchies places them into smaller groupings, known as collections, which are more easily browsed and manipulated than a flat namespace. However, hierarchies require categorization decisions that locate resources at a single location in the hierarchy, a drawback when a resource has multiple valid categories. For example, in a hierarchy of vehicle descriptions containing collections for cars and boats, a description of a combination car/boat vehicle could belong in either collection. Ideally, the description

should be accessible from both.

Hierarchies also make resource sharing more difficult, since resources that have utility across many collections are still forced into a single collection. For example, the mathematics department at one university might create a collection of information on fractals that contains bindings to some local resources, but also provides access to some resources at other universities. For many reasons, it may be

undesirable to make physical copies of the shared resources on the local server - to conserve disk space, to respect copyright constraints, or to make any changes in the shared resources visible automatically.

This protocol provides two mechanisms for allowing resources to appear in multiple places in an http URL hierarchy, and for sharing resources: bindings and redirect references.

The WebDAV Distributed Authoring Protocol added to the Web the ability to navigate Web resources hierarchically, complementing existing hypertext navigation facilities. In hypertext navigation, links appear in a specific order in a document. By contrast, hierarchical navigation has fewer mechanisms for expressing the ordering of a set of resources.

There are many scenarios where it is useful to impose an ordering on a collection, such as expressing a recommended access order, or a revision history order. Orderings may be based on property values, but they may be completely independent of any properties on the resources identified by the collection's internal member URIs. Orderings based on properties can be obtained using a search protocol [DASL], but orderings not based on properties need some other mechanism. These orderings generally need to be maintained by a human user. The ordering protocol defined here focuses on support for such human-maintained orderings, but also allows for server-maintained orderings.

4 Shared Resources

4.1 Overview

Shared resources make the same resource accessible from multiple locations in http URL namespaces. This protocol provides two mechanisms for sharing resources: bindings and redirect references.

The binding mechanism defined in this specification provides a way for clients to add new URI mappings to existing resources. A URI mapping is an association between an absolute URI and a resource, which makes it possible to submit requests to the resource using that URI as the Request-URI. Bindings, and the URI mappings based on them, are appealing mechanisms for resource sharing because:

- Once URI mappings are created with a BIND request, clients need do nothing special to use them. They behave just like any other URI mappings, transparently applying operations to the target resource.
- HTTP and WebDAV servers already provide URI mappings, so there is little extra work involved in allowing clients to create them.
- The integrity of bindings is guaranteed. MOVE and DELETE operations cannot leave bindings in an inconsistent state.

A limitation of bindings is that a server would need proxy capabilities in order to support bindings to resources on another server. In light of this complexity, support for cross-server bindings is OPTIONAL.

A redirect reference is a resource whose purpose is to provide access to another resource. It redirects most requests on the reference to another resource, thereby providing a form of mediated access to the other resource. Since the HTTP 302 (Moved Temporarily) status code is used to redirect client requests on a redirect reference, from the clientÆs point of view redirect references are less convenient to use than bindings. Redirect references require action by the client before they can be resolved. Moreover, the server is not required to enforce the integrity of redirect references. However, redirect references have a number of advantages:

- They are simple for servers to implement. Servers already provide redirect resources in the form of 301 / 302 redirection control, and the same mechanism can be used for redirect references.
- The same simple implementation works equally well for target resources that reside on the same server and for target resources that reside on a different server.
- Redirect references have only limited security implications.
- Since redirect references are resources, they can carry properties of their own.

Ideally, non-referencing clients should be able to use both bindings and redirect references. This goal is more difficult to meet for redirect references, since client action is required to resolve them. The strategy of having redirect references respond to most requests with a **302 (Moved Temporarily), accompanied by the URI of the target resource** in the Location header, fulfills this goal in most cases.

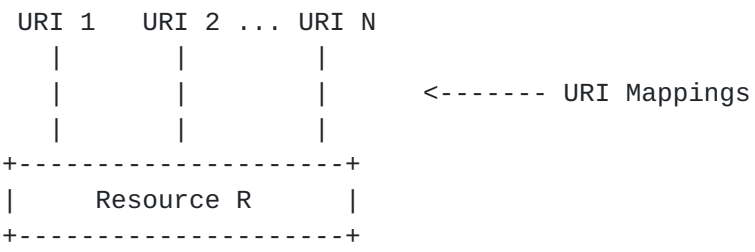
4.2 Bindings

Bindings are part of the state of a collection. In general, there is a one-to-one correspondence between a collection's bindings and its internal member URIs. The URI segment associated with a resource by one of a collection's bindings is also the final segment of one of the collection's internal member URIs. The final segment of each internal member URI identifies one of the bindings that is part of the collection's state, unless the internal member URI is not bound to a resource.

Even though a binding is just an association between a path segment and a resource, a binding creates one or more URI mappings of a URI to a resource. For example, if the segment "index.html" is being bound to a resource in a collection with URL "http://www.foo.net/A/", the binding creates a URI mapping of URL "http://www.foo.net/A/index.html" to the HTML resource. If the parent collection is then bound to the segment "B", this creates two URI mappings, "http://www.foo.net/B/" to the

collection resource, and "http://www.foo.net/B/index.html" to the HTML resource. Both the collection and the HTML resource are now accessible via two URLs apiece.

For a resource implemented by a computer, the relationship between a URI mapping and a resource is highlighted in the following diagram:



As discussed in [URI], a resource is an abstraction that maps a URI to an entity or set of entities. This resource can have multiple URIs/URLs mapped to it.

The identity of a binding is determined by the URI segment (in its collection) and the resource that the binding associates. If the resource is destroyed, the binding also goes out of existence. If the URI segment comes to be associated with a different resource, the original binding ceases to exist and another binding is created.

Bindings are not unique to advanced collections, although the BIND method for explicitly creating bindings is introduced here. Existing methods that create resources, such as PUT, MOVE, COPY, and MKCOL, implicitly create bindings. There is no difference between implicitly created bindings and bindings created with BIND.

Since a binding is an association between a path segment and a resource, it would be very undesirable if one binding could be destroyed as a side effect of operating on the resource through a different binding. It is not acceptable for a DELETE or MOVE through a different binding to destroy the resource or fail to update one binding, turning that binding into a dangling path segment. As a result, implementations MUST act to ensure the integrity of bindings.

It is especially difficult to maintain the integrity of cross-server bindings. Unless the server where the resource resides knows about all bindings on all servers to that resource, it may unwittingly destroy the resource or move it without notifying a server where a binding resides. For example, if server A permits creation of a binding to a resource on server B, server A must notify server B about its binding and must have an agreement with B that B will not destroy the resource while A's binding exists. Otherwise server B may receive a DELETE request that it

thinks removes the last binding to the resource and destroy the resource while A's binding still exists.

Consequently, support for cross-server bindings is OPTIONAL.

4.2.1 BIND Method

The BIND method creates a new binding from the final segment of the Request-URI (minus any trailing slash) to the resource identified by the Destination header. This binding is added to the collection identified by the Request-URI minus its trailing slash (if present) and final segment. The Destination header is defined in Section 9.3 of

[WebDAV].

If a server cannot guarantee the binding behavior specified for GET (Section 4.2.12), DELETE (Section 4.2.8), and MOVE (Section 4.2.10), the BIND request MUST fail with a 501 (Not Implemented) status code.

If the Request-URI ends in a slash ("/") (i.e., the Request-URI identifies a collection), the resource identified by the Destination header MUST be a collection resource, or the request fails with a 409 (Conflict) status code. This ensures that URIs ending in a slash are always bound to collections. If the Request-URI does not contain a path segment (i.e., it consists simply of a slash "/"), the BIND operation MUST fail and report a 409 (Conflict) status code.

After successful processing of a BIND request, it MUST be possible for clients to use the Request-URI to submit requests to the resource identified by the Destination header.

By default, if the Request-URI identifies an existing binding, the new binding replaces the existing binding. This default binding replacement behavior can be overridden using the Overwrite header defined in Section 9.6 of [WebDAV].

The Position request header (defined in Section 6.6) MAY be used in BIND requests.

A server MAY allow the BIND method to be used to create bindings to resources that support content negotiation or to resources that dynamically generate response entities.

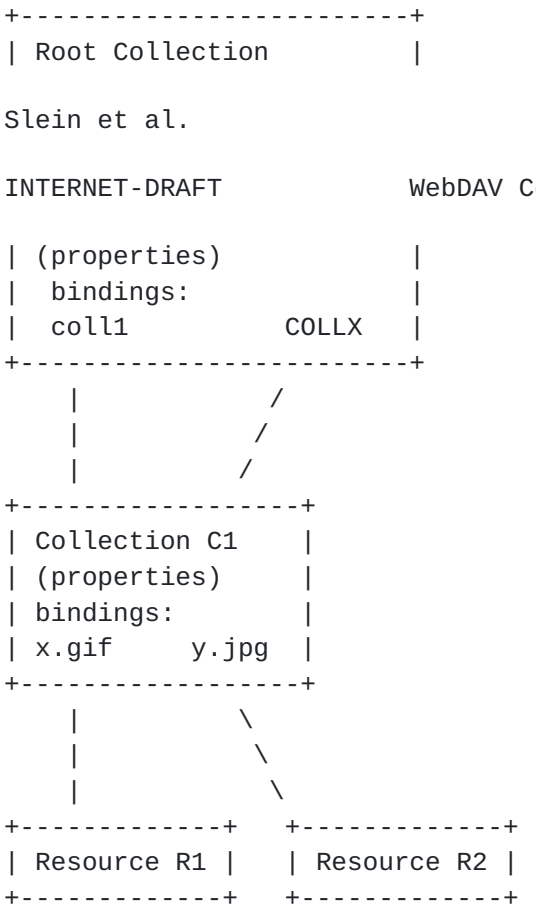
4.2.2 Bindings to Collections

BIND can create a binding to a collection resource. A collection accessed through such a binding behaves exactly as would a collection accessed through any other binding. Bindings to collections can result

in loops, which servers MUST detect when processing "Depth: infinity" requests. When a loop is detected, the server MUST respond with a 506 (Loop Detected) status code (defined in Section 7.1).

Creating a new binding to a collection makes each resource associated with a binding in that collection accessible via a new URI, but does not result in the creation of a new binding for each of these resources.

For example, suppose a new binding COLLX is created for collection C1 in the figure below. It immediately becomes possible to access resource R1 using the URI /COLLX/x.gif and to access resource R2 using the URI /COLLX/y.jpg, but no new bindings for these child resources were created. This is because bindings are part of the state of a collection, and associate a URI that is *relative to that collection* with its target resource. No change to the bindings in Collection C1 is needed to make its children accessible using /COLLX/x.gif and /COLLX/y.jpg.



4.2.3 URI Mappings Created by BIND

The set of URI mappings created by a successful BIND operation MUST be determined as follows:

1. Start with an empty set of URLs, called U.

2. Take the Request-URI and remove path segments (and associated "/" characters) from the right until either (a) a non-WebDAV collection, or a non-WebDAV advanced collection is found, or (b) the root, "/" is reached (i.e., no characters after the scheme and authority parts of the URL). This is the base URL B.
3. Add B, and all possible domain name variants of B (i.e., all other domain names which can be substituted for the domain name in B, and still retrieve the resource mapped to B), to URL set U.
4. Calculate the next path segment of the Request-URI, going from right to left, and call it S, which is bound to resource R.
5. For each member of URL set U, called Um, remove Um, then for every possible binding to R, create a new URL by adding the binding's path segment to Um, then add this new URL to U.
6. If there is no further path segment, then U has the complete set of URI mappings. Otherwise, go back to step 4.

4.2.4 Example: Generating the Set of URI Mappings

Assume a server responds to two domain names, www.fuzz.com, and fuzz.com, and has a top level that is not WebDAV-aware, called A/. Below A/ is an advanced collection that is bound to both 1/ and one/. In collection one/ there is a resource called index.html.

>> Request:

```
BIND /A/1/info.html HTTP/1.1
Host: www.fuzz.com
Destination: http://www.fuzz.com/A/one/index.html
```


>> Response:

```
HTTP/1.1 201 Created
```

The set of all possible URI mappings to the resource identified by http://www.fuzz.com/A/one/index.html is calculated as follows:

1. U is empty.
2. The base URL, B, is http://www.fuzz.com/A/, since A/ is not WebDAV- aware.
3. Since there are two domain names for this server, the domain name variations of B are added to U, making U contain: http://www.fuzz.com/A/ and http://fuzz.com/A/.
4. (iteration 1) The next path segment of the Request-URI is 1/, which is bound to an advanced collection resource, R.
5. (iteration 1) Since the advanced collection resource R is bound to 1/ and one/, the value of U after the operation is: http://www.fuzz.com/A/1/, http://www.fuzz.com/A/one/,

http://fuzz.com/A/1/, and http://fuzz.com/A/one/.
6. Go back to step 4, since there is one more path segment in the Request-URI.
4. (iteration 2) The next path segment of the Request-URI is info.html, which is bound to an HTML resource, R.
5. (iteration 2) Since the HTML resource is bound to info.html and index.html, the value of U after the operation is:
http://www.fuzz.com/A/1/index.html, http://www.fuzz.com/A/1/info.html, http://www.fuzz.com/A/one/index.html, http://www.fuzz.com/A/one/info.html, http://fuzz.com/A/1/index.html, http://fuzz.com/A/1/info.html, http://fuzz.com/A/one/index.html, http://fuzz.com/A/one/info.html.
6. Since there are no further path segments in the Request-URI, U now has the complete set of URI mappings for the resource identified by the Destination header.

4.2.5 Status Codes

201 (Created): The binding was successfully created.

400 (Bad Request): The client set an invalid value for the Destination or Position header.

409 (Conflict): Several conditions may produce this response. The URI in the Destination header is not mapped to a resource. The request is attempting to create a binding in a collection that does not exist. The request is attempting to position the binding in an unordered collection. The request is attempting to re-bind the top-level collection.

412 (Precondition Failed): The Overwrite header is "F", and a binding already exists for the request-URI.

4.2.6 Example: BIND

>> Request:


```
BIND /~whitehead/dav/spec08.txt HTTP/1.1
Host: www.ics.uci.edu
Destination: http://www.ics.uci.edu/pub/i-d/draft-webdav-protocol-08.txt
```

>> Response:

```
HTTP/1.1 201 Created
```

The server created a new binding, associating "spec08.txt" with the resource identified by the URL "http://www.ics.uci.edu/pub/i-d/draft-

webdav-protocol-08.txt". Clients can now use the Request-URI, "http://www.ics.uci.edu/~whitehead/dav/spec08.txt", to submit requests to that resource. As part of this operation, the server added the binding "spec08.txt" to collection /~whitehead/dav/.

4.2.7 Example: BIND Conflict

>> Request:

```
BIND /press/prlogo.gif HTTP/1.1
Host: www.softcorp.com
Destination: http://www.softcorp.com/logos/
```

>> Response:

```
HTTP/1.1 409 Conflict
```

The client requested the server to create a binding between "prlogo.gif" and the resource identified by the URL "http://www.softcorp.com/logos/". Since the Destination does end in a slash, while the Request-URI does not, the server failed the request, returning a 409 (Conflict) status code.

4.2.8 DELETE and Bindings

The DELETE method requests that the server remove the binding between the resource identified by the Request-URI and the binding name, the last path segment of the Request-URI. The binding MUST be removed from its parent collection, identified by the Request-URI minus its trailing slash (if present) and final segment. The All-Bindings header may be used with DELETE to request that the server remove all bindings to the resource identified by the Request-URI.

Once all bindings to the resource are removed, the server MAY reclaim system resources associated with the resource. If DELETE removes a binding to a resource, but there remain other bindings to that resource, the server MUST NOT reclaim system resources associated with the resource.

Since DELETE as specified in [WebDAV] is not an atomic operation, it may happen that parts of the hierarchy under the request-URI cannot be deleted. In this case, the response is as described in [WebDAV].

[HTTP] states that "the DELETE method requests that the origin server delete the resource identified by the Request-URI." Because [HTTP] did

not distinguish between bindings and resources, the intent of its definition of DELETE is unclear. We consider the definition presented

here to be a clarification of the definition in [HTTP].

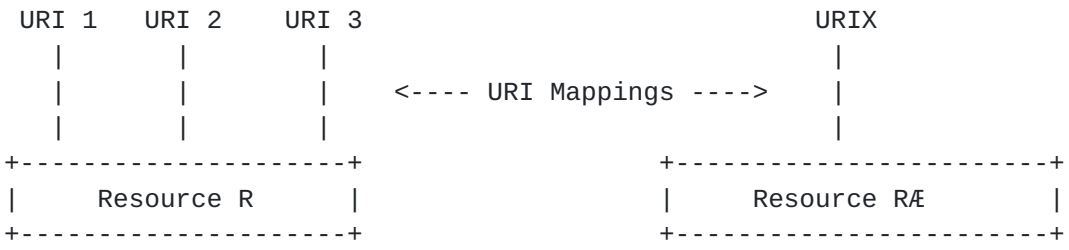
Section 8.6.1 of [WebDAV] states that during DELETE processing, a server "MUST remove any URI for the resource identified by the Request-URI from collections which contain it as a member." Servers that support bindings SHOULD NOT follow this requirement.

4.2.9 COPY and Bindings

As defined in Section 8.8 of [WebDAV], COPY causes the resource identified by the Request-URI to be duplicated, and makes the new resource accessible using the URI specified in the Destination header. Upon successful completion of a COPY, a new binding is created between the last path segment of the Destination header (including trailing "/", if present), and the destination resource. The new binding is added to its parent collection, identified by the Destination header minus its trailing slash (if present) and final segment.

A COPY with "Depth: 0" MUST NOT duplicate the bindings contained by the collection.

As an example, suppose that a COPY is issued to URI 3 for resource R below (which is also mapped to URI 1 and URI 2), with the Destination header set to URIX. After successful completion of the COPY operation, resource R is duplicated to create resource R', and a new binding has been created which creates at least the URI mapping between URIX and the new resource (although other URI mappings may also have been created).



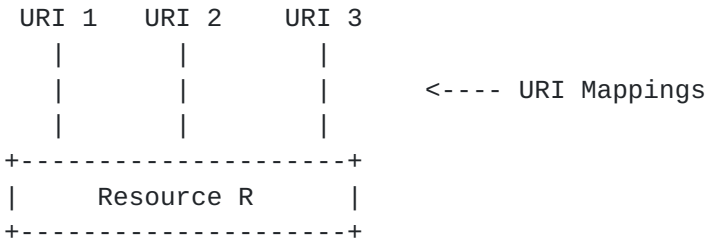
4.2.10 MOVE and Bindings

The MOVE method has the effect of creating a new binding to a resource (at the Destination), and removing an existing binding (at the Request-URI). The name of the new binding is the last path segment of the Destination header, and the new binding is added to its parent collection, identified by the Destination header minus its trailing slash (if present) and final segment.

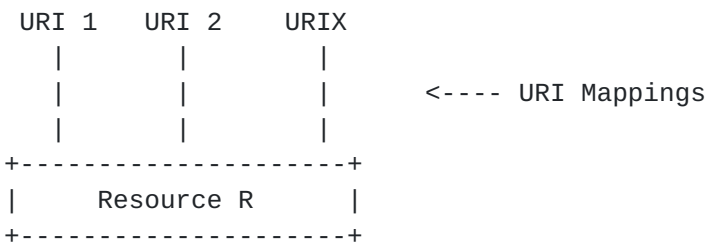
As an example, suppose that a MOVE is issued to URI 3 for resource R below (which is also mapped to URI 1 and URI 2), with the Destination header set to URIX. After successful completion of the MOVE operation, a new binding has been created which creates at least the URI mapping between URIX and resource R (although other URI mappings may also have been created). The binding corresponding to the final segment of URI 3

has been removed, which also causes the URI mapping between URI 3 and R to be removed.

>> Before Request:



>> After Request:



Since MOVE as specified in [WebDAV] is not an atomic operation, it may happen that parts of the hierarchy under the request-URI can be moved. In this case, the response is as described in [WebDAV].

4.2.10.1 Implementation Note

In some situations, particularly when the destination is on a different server from the original resource, the server may implement MOVE by performing a COPY, performing some consistency maintenance on bindings and properties, and then performing a DELETE. In the end, all of the original bindings except the one corresponding to the Request-URI will be associated with the new resource. The binding corresponding to the URI in the Destination header will be associated with the new resource. And the original resource together with the binding corresponding to the Request-URI will have been deleted. This implementation is in accord with the definition of MOVE in [WebDAV], and is logically equivalent to the definition given above.

The consistency maintenance processing that is required for this implementation is as follows:

The DAV:creationdate property of the new resource SHOULD have the same value as the DAV:creationdate property of the old resource.

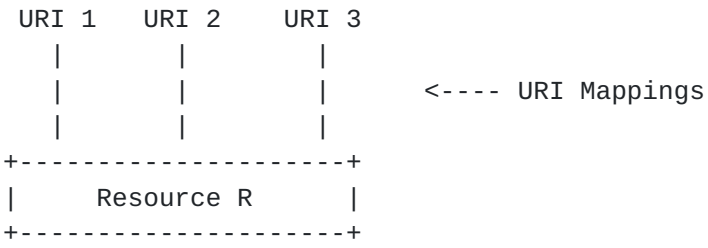
The DAV:getlastmodified property of the new resource SHOULD have the same value as the DAV:getlastmodified property of the old resource.

All URIs that were bound to the original resource except for the Request-URI MUST be bound instead to the new resource.

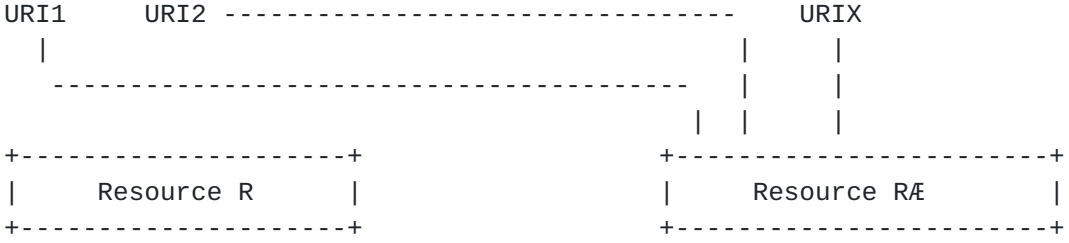
Consider again the case where a MOVE is issued to URI 3 for resource R (which is also mapped to URI 1 and URI 2), with the Destination header set to URIX. Unlike the previous example, in this implementation, after successful completion of the MOVE operation, resource R has been

duplicated as resource R'. The original bindings corresponding to URI 1 and URI2 are now associated with R'. The binding corresponding to the Request-URI (URI 3) has been removed. And a new binding has been created which creates at least the URI mapping between URIX and resource R'. Note that the server may reclaim the storage associated with resource R once the MOVE operation has finished.

>> Before Request:



>> After Request:



4.2.11 LOCK and UNLOCK

Bindings do not affect the semantics of locks, as specified in [WebDAV]. Specifically, the requirement in section 8.10.3 that "a LOCK request on a resource MUST NOT succeed if it can not be honored by all the URIs through which the resource is accessible" still holds. The LOCK method locks the resource, and a lock is visible via all URIs mapped to that resource. Similarly, a successful UNLOCK issued via any URI mapping to a resource removes the lock from the resource, and this lock removal is visible via all URI mappings.

When a resource is locked, the lock owner expects to be able to access the resource -- using the same Request-URI that he used to lock the resource -- for as long as he holds the lock. This would not be possible if another user could move or delete any of the collections corresponding to segments of the request-URI.

Consequently, for the duration of a lock, it MUST NOT be possible for a principal other than the lock owner to make a locked resource inaccessible via the URI mapping used to lock the resource. Only the lock owner can move or delete any of the collections corresponding to segments of the Request-URI. This restriction does not prevent others from modifying those collections, by adding members to them, removing members from them, or changing their property values.

For example, if a user locks /plants/herbs/rosemary.html, it is not possible for another user to move /plants/herbs/ to /plants/flowering/herbs/, or to completely delete /plants/herbs/, though

it is possible this delete operation may succeed in deleting everything except for /plants/herbs/rosemary.html and /plants/herbs/.

4.2.12 Bindings and Other Methods

This section describes the interaction of bindings with those HTTP methods not yet explicitly discussed. The semantics of the methods GET, HEAD, PUT, POST and OPTIONS are specified in [HTTP]. The semantics of PROPFIND, PROPPATCH, and MKCOL are specified in [WebDAV].

For most of these methods, no new complexities are introduced by allowing explicit creation of multiple bindings to the same resource. For the access operations (GET, HEAD, OPTIONS, and PROPFIND), it is simply the case that no matter which URI mapping to a given resource is used as the Request-URI, the response is mediated by that same resource. The responses may, however, vary depending upon which Request-URI was used. For example, the response to a GET request may contain the Request-URI in its entity.

The same is true for POST. No matter which URI mapping to a given resource is used as the Request-URI, the response is mediated by that same resource. The changes made on the server and the responses may, however, vary depending upon which Request-URI was used.

If the Request-URI of a PUT identifies an existing resource, then a PUT via one URI mapping to this resource MUST produce the same result as a PUT with the same headers and request entity body via any other URI mapping to the same resource. The change made by a PUT via one URI mapping MUST affect the resource that generates the GET response for all

URI mappings to the same resource.

A PROPPATCH through one URI mapping to a resource MUST produce the same changes to its properties as the same PROPPATCH request through a different URI mapping to the same resource.

As specified in [WebDAV], MKCOL cannot overwrite an existing resource. MKCOL through any URI mapping to an existing resource must fail.

The semantics of MKREF are specified in Section 4.5.1 below. A MKREF through one URI mapping to a resource MUST produce the same result as a MKREF with the same headers through any other URI mapping to the same resource. By default, it overwrites the resource with a new redirect reference.

The semantics of ORDERPATCH are specified in 5.5.1 below. An ORDERPATCH through one URI mapping to a collection MUST produce the same changes to its ordering as the same ORDERPATCH request through any other URI mapping to the same collection.

4.2.13 Discovering the Bindings to a Resource

An OPTIONAL DAV:bindings property on a resource provides a list of the bindings that associate URI segments with that resource. By retrieving this property, a client can discover the bindings that point to the resource and the collections that contain bindings to the resource. As

for all DAV: properties, this specification is silent as to how the DAV:bindings property is implemented on the server.

Rationale: A number of scenarios require clients to navigate from a resource to the bindings that point to it, and to the collections that contain those bindings. This capability is particularly important for Document Management Systems. Their clients may need to determine, for any object in the DMS, what collections contain bindings to that object. This information can be used for upward navigation through a hierarchy or to discover related documents in other collections.

Risks: When deciding whether to support the DAV:bindings property, server implementers / administrators should balance the benefits it provides against the cost of maintaining the property and the security risks enumerated in Sections 12.5 and 12.6.

4.3 Redirect References

For most operations submitted to a redirect reference, the response is a **302 (Moved Temporarily), accompanied by the Resource-Type header** (defined in Section 6.2 below) set to "DAV:redirectref" and the Location

header set to the URI of the target resource. With this information, the client can resubmit the request to the URI of the target resource. The methods COPY (for collections containing redirect references), DELETE, MOVE, and LOCK, for reasons that will be explained, are exceptions to this general behavior. These exceptional operations are applied to the reference itself and do not result in a 302 response.

If the client is aware that it is operating on a redirect reference, it can resolve the reference by retrieving the reference's DAV:reftarget property (defined in Section 7.1 below), whose value is the URI of the target resource. It can then submit requests to the target resource.

A redirect reference is a new type of resource. To distinguish redirect references from ordinary resources, a new value of the DAV:resourcetype property (defined in [WebDAV]), DAV:redirectref, is defined in Section **8.1 below.**

Since a redirect reference is a resource, it is possible to apply methods to the reference rather than to its target. The Passthrough request header (defined in Section 6.4 below) is provided so that referencing-aware clients can control whether an operation is applied to the redirect reference or to its target resource. The Passthrough header can be used with most requests to redirect references. This header is particularly useful with PROPFIND, to retrieve the reference's own properties.

4.3.1 MKREF Method

The MKREF method creates a redirect reference resource identified by the Request-URI, whose target is identified by the REQUIRED Ref-Target header. MKREF sets the value of the REQUIRED DAV:reftarget property to the value of the Ref-Target header.

The MKREF method creates a new binding between the new redirect

reference resource and the last path segment of the Request-URI. The new binding is added to its parent collection, identified by the Request-URI minus its trailing slash (if present) and final segment.

The Position request header (defined in Section 6.6) MAY be used in MKREF requests.

MKREF requests MAY include an entity body. This specification does not define the action to be taken if a request entity body is present, but allows it for extensibility.

By default, if the Request-URI of the MKREF request identifies an existing resource, the server MUST perform a delete operation on the

existing resource before performing the MKREF. This default behavior can be overridden using the Overwrite header defined in Section 9.6 of [WebDAV].

4.3.1.1 Status Codes

201 (Created): The redirect reference resource was successfully created.

400 (Bad Request): The client set an invalid value for the Ref-Target or Position header.

409 (Conflict): Several conditions may produce this response. There may be no resource at the location specified in Ref-Target, on a server that prohibits dangling references. The request may be attempting to create the reference in a collection that does not exist. The request may be attempting to position the reference before or after a resource that is not in the collection, or before or after itself. The request may be attempting to position the reference in an unordered collection.

412 (Precondition Failed): The Overwrite header is "F", and a resource already exists at the request-URI.

4.3.1.2 Example: MKREF

>> Request:

```
MKREF /~whitehead/dav/spec08.ref HTTP/1.1
Host: www.ics.uci.edu
Ref-Target: /i-d/draft-webdav-protocol-08.txt
```

>> Response:

```
HTTP/1.1 201 Created
```

This request resulted in the creation of a new redirect reference at www.ics.uci.edu/~whitehead/dav/spec08.ref, which points to the resource identified by the Ref-Target header. In this example, the target resource of the referential resource is identified by the URI http://www.ics.uci.edu/~whitehead/dav/i-d/draft-webdav-protocol-08.txt. The referential resource's DAV:resourcetype property is set to DAV:redirectref. Its DAV:reftarget property is set to the value of the Ref-Target header, "/i-d/draft-webdav-protocol-08.txt".

4.3.2 Listing the Redirect References in a Collection

A URI of a redirect reference can be an internal member URI of a collection just as the URI of an ordinary resource can. A listing of

the internal member URIs of a collection shows all of the URIs that are internal members of the collection, whether they identify redirect references or ordinary resources. That is, a WebDAV PROPFIND request on a collection resource with the Depth header set to 1 or infinity MUST return a response XML element for each member URI in the collection, whether it identifies an ordinary resource or a redirect reference.

For each redirect reference, the response element MUST contain a 302 (Moved Temporarily) status code unless a Passthrough header with the value "F" is included with the PROPFIND request. The DAV:location and DAV:resourcetype properties MUST be included with the 302 status code, extending the syntax of the DAV:response element that was defined in [WebDAV] as described in Section 9 below. A referencing-aware client can tell from the DAV:resourcetype property that the collection contains a redirect reference. The DAV:location property contains the absolute URI of the target resource. A referencing-aware client can either use the URI value of the DAV:location property to retrieve the properties of the target resource, or it can submit a PROPFIND to the redirect reference with "Passthrough: F" to retrieve its properties. It is recommended that future editors of [WebDAV] define the DAV:location property in [WebDAV], so that non-referencing clients will also be able to use the response to retrieve the properties of the target resource.

If the Depth header is set to infinity in the PROPFIND request, the server MUST NOT follow redirect references into any collections to which they may refer.

The Passthrough header (defined in Section 6.4) MAY be used with a PROPFIND request on a collection.

4.3.2.1 Example: PROPFIND on a Collection with Redirect References

Suppose a PROPFIND request with Depth = infinity is submitted to the following collection, with the members shown here:

http://www.svr.com/MyCollection/
   (ordinary resource) diary.html
   (redirect reference) nunavut

>> Request:

```
PROPFIND /MyCollection/ HTTP/1.1
Host: www.svr.com
Depth: infinity
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:keywords/>
   </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>
            <J:keywords>diary, interests, hobbies</J:keywords>
         </D:prop>
         <D:status>HTTP/1.1 200 OK</D:status>
      </D:propstat>
   </D:response>
   <D:response>
      <D:href>http://www.svr.com/MyCollection/diary.html</D:href>
      <D:propstat>
         <D:prop>
            <D:resourcetype/>
            <J:keywords>diary, travel, family, history</J:keywords>
         </D:prop>
         <D:status>HTTP/1.1 200 OK</D:status>
      </D:propstat>
   </D:response>
   <D:response>
      <D:href>http://www.svr.com/MyCollection/nunavut</D:href>
      <D:status>HTTP/1.1 302 Moved Temporarily</D:status>
      <D:prop>
         <D:location>
            <D:href>http://www.inac.gc.ca/art/inuit/</D:href>
         </D:location>
         <D:resourcetype><D:redirectref/></D:resourcetype>
      </D:prop>
   </D:response>
</D:multistatus>
```

In this example the Depth header is set to infinity, and the Passthrough header is not used. The collection contains one URI that identifies a

redirect reference. The response element for the redirect reference has a status of 302 (Moved Temporarily), and includes a DAV:prop element with the DAV:location and DAV:resourcetype properties to allow clients to retrieve the properties of its target resource. (The response element for the redirect reference does not include the requested properties. The client can submit another PROPFIND request to the URI in the DAV:location property to retrieve those properties.)

4.3.2.2 Example: PROPFIND with Passthrough: F on a Collection with Redirect References

Suppose a PROPFIND request with Passthrough = F and Depth = infinity is submitted to the following collection, with the members shown here:

```
/MyCollection/
   (ordinary resource) diary.html
   (redirect reference) nunavut
```

>> Request:

```
PROPFIND /MyCollection/ HTTP/1.1
Host: www.svr.com
Depth: infinity
Passthrough: F
Content-Type: text/xml
Content-Length: xxxx

<?xml version="1.0" ?>
<D:propfind xmlns:D="DAV:">
   <D:prop>
      <D:resourcetype/>
      <D:reftarget/>
   </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:">
   <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> <D:reftarget/> </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/diary.html</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> <D:reftarget/> </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/nunavut</D:href>
      <D:propstat>
         <D:prop>
            <D:resourcetype><D:redirectref/></D:resourcetype>
            <D:reftarget>
               <D:href>http://www.inac.gc.ca/art/inuit/</D:href>
            </D:reftarget>
         </D:prop>
         <D:status>HTTP/1.1 200 OK</D:status>
      </D:propstat>
   </D:response>
</D:multistatus>
```

Since the Passthrough header has the value "F", the response shows the properties of the redirect reference in the collection rather than the properties of its target. The value of the Passthrough header also prevents a 302 response from being returned for the redirect reference.

4.3.3 Copying Redirect References

A client's intent in performing a COPY operation is to create a new resource that is similar to the original resource and behaves like the original resource, and that can be modified without affecting the

original resource. For a COPY request to a redirect reference, the expectation would be a 302 response that the client could use to copy the target resource. This would yield an independent resource that could be modified without affecting the original resource. For COPY requests to collections that contain redirect references, the situation is less clear. There is tension between two expectations. On the one hand, the client may expect the new copy of the collection to behave like the old one (which implies having references where the old one had references). On the other hand, the client may expect that it will be possible to modify the resources in the new collection without affecting the resources in the old collection (which implies having copies of the targets where the original collection had references).

For a COPY request on an individual reference, the response MUST be a **302 (Moved Temporarily) status code, with the URI of the target resource** in the Location header, and "Resource-Type: DAV:redirectref" to distinguish the response from an ordinary HTTP redirect. This is the normal behavior for redirect references, allowing the client to resubmit the request to the target resource identified in the Location header. This also yields intuitively correct behavior for a COPY request to an individual reference. Reference-aware clients can use the Passthrough header with the value "F" to copy the redirect reference itself.

For COPY on a collection containing redirect references, different

semantics may be desirable in different scenarios. Consequently, this specification makes a fairly arbitrary choice to take the simplest path. When a COPY request is submitted to a collection containing redirect references, the server MUST copy the redirect references to the new collection rather than returning 302 status codes for them. This will result in a new collection that behaves like the old one, and avoids responding with multiple 302 status codes, each of which the client would have to process separately. Reference-aware clients can force the server to respond with 302 status codes rather than copying the references by using the Passthrough header with the value "T".

4.3.3.1 Example: COPY on a Redirect Reference

>> Request:

```
COPY /MyCollection/tuva HTTP/1.1
Host: www.svr.com
Destination: http://www.svr.com/OtherCollection/tuva.html
```

>> Response:

```
HTTP/1.1 302 Moved Temporarily
Location: http://www.svr.com/Asia/History/tuva.html
```

Resource-Type: DAV:redirectref

In this example, the request-URI identifies a redirect reference whose target resource is identified by http://www.svr.com/Asia/History/tuva.html. In this case, the server responded with a 302, and provided the URL of the target resource in the Location header. The Resource-Type header indicates to a reference-aware client that this is not an HTTP 1.1 redirect, but a reference to the resource identified by the Location header. The client can now resubmit the COPY request to the target resource, producing the desired result: a duplicate of the original target resource that can be modified independently of the original.

4.3.3.2 Example: COPY on a Collection That Contains a Redirect Reference

Suppose a COPY request is submitted to the following collection, with the members shown:

```
/MyCollection/
     (ordinary resource) diary.html
     (redirect reference) nunavut with target /Someplace/nunavut.map

>> Request:

COPY /MyCollection/ HTTP/1.1
Host: www.svr.com
Destination: http://www.svr.com/OtherCollection/

>> Response:

HTTP/1.1 201 Created
```


In this case, since /MyCollection/nunavut is a redirect reference, the reference itself, and not its target, was copied into the new collection. So the resulting collection is as follows:

```
/OtherCollection/
      (ordinary resource) diary.html
      (redirect reference) nunavut with target /Someplace/nunavut.map
```

4.3.4 Deleting and Moving Redirect References

The DELETE method is used to delete bindings to redirect references. DELETE MUST affect bindings to the reference itself, unless "Passthrough: T" is used, in which case it generates a 302 (Moved Temporarily) response. Similarly, when a DELETE on a collection encounters a redirect reference in the subtree under that collection, it

MUST delete bindings to the reference, unless "Passthrough: T" is used, in which case it generates a 302 (Moved Temporarily) response. Whether deleting an individual resource or a collection, DELETE on a redirect reference does not affect the target of the reference.

A MOVE operation on a redirect reference MUST move the reference to a different location, and MUST NOT change the location of its target, unless "Passthrough: T" is used, in which case a 302 (Moved Temporarily) response is generated. The DAV:reftarget property is unchanged after a MOVE. Similarly, when a MOVE on a collection encounters a redirect reference in the subtree under that collection, it MUST move the reference, and not its target, unless "Passthrough: T" is used, in which case a 302 (Moved Temporarily) response is generated.

DELETE and MOVE differ from other methods in that they do not alter the resource that is being deleted or moved, but rather the collection that contains its binding. They change the membership of that collection.

When a redirect reference is added to a collection, the aim is to make it look as if the target resource were a member of that collection. When the reference is removed from that collection, the aim is to change the membership of that collection. Membership of the target in any other collections, either internally or by reference, should not be affected. Consequently, DELETE and MOVE do not follow the normal rules of behavior for references. Instead, they are applied by default to the reference itself, not to its target, and by default do not result in 302 status codes.

4.3.5 Locking Redirect References

The semantics of LOCK described here resulted from balancing a set of incompatible considerations:

- Ideally, a LOCK on a redirect reference should lock both the reference and its target resource. The owner of an exclusive write lock, for example, would be surprised if anyone else could modify the content of the target resource while he held the lock. He would also be surprised if anyone else could delete the reference to it, or replace the reference with one pointing to a different target.
- Non-referencing clients should be able to use redirect references without encountering surprising results.
- The basic characteristics of redirect references should be honored. Redirect references should be simple for servers to implement. In particular, a server should never have to resolve a redirect reference. A server should not have to provide proxy capabilities in order to implement redirect references.

- There should be consistency between the behavior of LOCK on a single redirect reference and the behavior of LOCK on a collection that contains redirect references.
- The behavior of all requests to redirect references should be as consistent as possible. In the absence of a Passthrough header, all methods should return a 302 when sent to a redirect reference.
- LOCK semantics for redirect references should be consistent with the LOCK semantics defined in [WebDAV].

We have compromised the intuitive locking behavior and support for non-referencing clients in order to preserve various sorts of consistency.

4.3.6 LOCK on Redirect References

The behavior of LOCK for redirect references was determined by what is possible for the case of locking collections that contain redirect references.

The default behavior for any operation on a redirect reference is that a **302 (Moved Temporarily) response will be returned, unless the** Passthrough header with a value of "F" is used. However, this policy has unacceptable consequences when locking a collection that contains redirect references. Since [WebDAV] requires LOCK on a collection to be an atomic operation, if a 302 response is received for any member of the collection, the entire LOCK must fail. This would make it impossible to lock any collection that contained a redirect reference.

To avoid this result, a LOCK with Depth > 0 on a collection MUST lock any redirect references it encounters, and not return 302 responses for them, unless the Passthrough header with a value of "T" is used. Use of the Passthrough header with a value of "T" in a LOCK request on a collection will cause the entire lock to fail if a redirect reference is encountered.

This gives part of the expected default lock behavior without forcing the server to resolve the redirect reference or become a proxy server in cases where the target resides on a different server.

There will be no hint in any response code that there are redirect references whose targets need to be locked. The client will most likely not lock any targets until it attempts an operation on the target and gets a 302 response. It is possible that a non-referencing client may never realize that the reference's target has not been locked.

Clearly, a LOCK with Depth = infinity on a collection MUST NOT follow any redirect references whose targets are collections into the target collections; it MUST NOT cause any resources in those target collections to be locked.

The behavior of LOCK for individual redirect references is designed to be consistent with LOCK behavior for collections that contain redirect references. By default a LOCK on a redirect reference MUST lock only the reference, not its target, and it MUST NOT return a 302 response. A reference-aware client can use the Passthrough header with a value of "T" to get a 302 response with the URI of the target resource in the Location header.

UNLOCK behaves as specified in [WebDAV], unlocking all resources included in the lock identified by the Lock-Token header.

4.3.6.1 Example: LOCK on a Redirect Reference

```
>> Request:

LOCK /MyCollection/tuva HTTP/1.1
Host: www.svr.com
Content-Type: text/xml
Content-Length: nnnn
Authorizaton: Digest username="jas",
   realm=jas@webdav.sb.aol.com, nonce=". . . ",
   uri="/MyCollection/tuva",
   response=". . . ", opaque=". . . "

<?xml version="1.0" ?>
<D:lockinfo xmlns:D="DAV:">
   <D:lockscope><D:exclusive/></D:lockscope>
   <D:locktype><D:write/></D:locktype>
   <D:owner>
      <D:href>http://www.svr.com/~jas/contact.html</D:href>
   </D:owner>
</D:lockinfo>

>> Response:

HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: nnnn

<?xml version="1.0" ?>
<D:prop xmlns:D="DAV:">
   <D:lockdiscovery>
      <D:activelock>
         <D:lockscope><D:exclusive/></D:lockscope>
         <D:locktype><D:write/></D:locktype>
         <D:depth>0</D:depth>
         <D:owner>
            <D:href>http://www.svr.com/~jas/contact.html</D:href>
         </D:owner>
```

```
         <D:locktoken>
            opaquelocktoken:e71dfae-5dec-22d6-fea5-00a0c91e6be4
         </D:locktoken>
      </D:activelock>
   </D:lockdiscovery>
</D:prop>
```


The request and response look exactly as specified in [WebDAV]. In this example, the request-URI, http://www.svr.com/MyCollection/tuva, identifies a redirect reference, which was successfully locked. The target resource of the redirect reference is not locked.

4.3.6.2 Example: LOCK on a Collection That Contains a Redirect Reference, with Passthrough: T

Suppose a LOCK request is submitted to the following collection, with the members shown:

```
/MyCollection/
     (ordinary resource) diary.html
     (redirect reference) nunavut

>> Request:

LOCK /MyCollection/ HTTP/1.1
Host: www.svr.com
Passthrough: T
Content-Type: text/xml
Content-Length: nnnn
Authorizaton: Digest username="jas",
   realm=jas@webdav.sb.aol.com, nonce=". . . ",
   uri="/MyCollection/tuva",
   response=". . . ", opaque=". . . "

<?xml version="1.0" ?>
<D:lockinfo xmlns:D="DAV:">
   <D:lockscope><D:exclusive/></D:lockscope>
   <D:locktype><D:write/></D:locktype>
   <D:owner>
      <D:href>http://www.svr.com/~jas/contact.html</D:href>
   </D:owner>
</D:lockinfo>

>> Response:

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

```
Content-Length: nnnn

<?xml version="1.0" ?>
<D:multistatus xmlns:D="Dav:">
   <D:response>
      <D:href>http://www.svr.com/MyCollection/</D:href>
      <D:propstat>
         <D:prop><D:lockdiscovery/></D:prop>
         <D:status>HTTP/1.1 424 Failed Dependency</D:status>
      </D:propstat>
   </D:response>
   <D:response>
      <D:href>http://www.svr.com/MyCollection/diary.html</D:href>
      <D:status>HTTP/1.1 424 Failed Dependency</D:status>
```



```
   </D:response>
   <D:response>
      <D:href>http://www.svr.com/MyCollection/nunavut</D:href>
      <D:status>HTTP/1.1 302 Moved Temporarily</D:status>
      <D:prop>
         <D:location>
            <D:href>http://www.inac.gc.ca/art/inuit/</D:href>
         </D:location>
         <D:resourcetype><D:redirectref/></D:resourcetype>
      </D:prop>
   </D:response>
</D:multistatus>
```

The "Passthrough: T" header caused the server to return a 302 response code for the redirect reference in the collection. Consequently, neither the collection nor any of the resources identified by its internal member URIs were locked. A referencing-aware client can submit a separate LOCK request to the URI in the DAV:location property returned for the redirect reference, and can resubmit the LOCK request with "Passthrough: F" to the collection. At that point both the reference and its target will be locked (as well as the collection and all the resources identified by its other members).

4.3.7 Other Operations on Redirect References

Although non-referencing-aware clients cannot create referential resources, they should be able to use the references created by reference-aware WebDAV clients. They should be able to follow any references to their targets. To make this possible, a server that receives a GET, HEAD, PUT, POST, OPTIONS, PROPFIND, PROPPATCH, MKCOL, MKREF, BIND, or ORDERPATCH request made via a redirect reference MUST return a 302 (Moved Temporarily) status code. The client and server MUST

follow [HTTP] Section 10.3.3 "302 Moved Temporarily," but with these additional rules:

o The Location response header MUST contain the absolute target URI of the reference.

o The response MUST include the Resource-Type header. This header allows reference-aware WebDAV clients to recognize the resource as a reference and understand the reason for the redirection.

A reference-aware WebDAV client can act on this response in one of two ways. It can, like a non-referencing client, resubmit the request to the URI in the Location header in order to operate on the target resource. Alternatively, it can resubmit the request to the URI of the redirect reference with the Passthrough header set to "F" in order to operate on the reference itself. If the Passthrough header is present with a value of "F", the request MUST be applied to the reference itself, and a 302 response MUST NOT be returned.

If a reference-aware client knows before submitting its request that the request-URI identifies a redirect reference, and if the client wants to apply the method to the reference, it can save the round trip caused by the 302 response by using "Passthrough: F" in its initial request to the

URI.

"Passthrough: F" can be used with GET or HEAD to retrieve the entity headers of a redirect reference. When "Passthrough: F" is used with GET or HEAD, the referencing entity headers (Ref-Type and Ref-Target) MUST be returned, along with all HTTP headers that make sense for references (at a minimum, Cache-Control, Age, ETag, Expires, and Last-Modified).

"Passthrough: F" can be used with PUT to replace the redirect reference with an ordinary resource. It can be used with OPTIONS to retrieve the capabilities of a redirect reference.

Clients MUST NOT, however, use "Passthrough: F" with POST. Since a reference cannot accept another entity as its subordinate, an attempt to POST to a reference with "Passthrough: F" will also fail. If a server receives a POST request with "Passthrough: F" on a redirect reference, it MUST fail the request with a 400 (Bad Request) status code.

Since MKCOL fails when applied to existing resources, if the client attempts to resubmit the request to the target resource, the request MUST fail (unless the reference is a dangling reference). Similarly, if the client attempts to resubmit the request to the reference with "Passthrough: F", the request MUST fail.

Since ORDERPATCH applies only to collections, an ORDERPATCH request with a Passthrough header with the value "F" on a redirect reference MUST fail.

4.3.7.1 Example: GET on a Redirect Reference

>> Request:

```
GET /bar.html HTTP/1.1
Host: www.foo.com
```

>> Response:

```
HTTP/1.1 302 Moved Temporarily
Location: http://www.svr.com/Internet/xxspec08.html
Resource-Type: DAV:redirectref
```

Since /bar.html is a redirect reference and the Passthrough header is not included in the request, the response is a 302 (Moved Temporarily). The Resource-Type header informs a reference-aware client that this is not an ordinary HTTP 1.1 redirect, but is a redirect reference. The URI of the target resource is provided in the Location header so that the client can resubmit the request to the target resource.

4.3.7.2 Example: PUT on a Redirect Reference with "Passthrough: F"

>> Request:

```
PUT /bar.html HTTP/1.1
Host: www.foo.com
Passthrough: F
```



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

. . . some content . . .
```

>> Response:

```
HTTP/1.1 200 OK
```

Although /bar.html is a redirect reference, the presence of the "Passthrough: F" header prevents a 302 response, and instead causes the request to be applied to the reference. The result in this case is that the reference is replaced by an ordinary resource having the content submitted with the request.

4.3.7.3 Example: PROPPATCH on a Redirect Reference

Request:

```
PROPPATCH /bar.html HTTP/1.1
Host: www.foo.com
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

   <?xml version="1.0" encoding="utf-8" ?>
   <D:propertyupdate xmlns:D="DAV:"
   xmlns:Z="http://www.w3.com/standards/z39.50/">
     <D:set>
          <D:prop>
               <Z:authors>
                    <Z:Author>Jim Whitehead</Z:Author>
                    <Z:Author>Roy Fielding</Z:Author>
               </Z:authors>
          </D:prop>
     </D:set>
     <D:remove>
          <D:prop><Z:Copyright-Owner/></D:prop>
     </D:remove>
   </D:propertyupdate>
```

Response:

```
HTTP/1.1 302 Moved Temporarily
Location: http://www.svr.com/Internet/xxspec08.html
Resource-Type: DAV:redirectref
```

Since /bar.html is a redirect reference and the Passthrough header is not included in the request, the response is a 302 (Moved Temporarily). The Resource-Type header informs a reference-aware client that this is not an ordinary HTTP 1.1 redirect, but is a redirect reference. The URI of the target resource is provided in the Location header so that the client can resubmit the request to the target resource.

4.3.8 Operations on Targets of Redirect References

Operations on targets of redirect references have no effect on the reference.

4.3.9 Relative URIs in Ref-Target and DAV:reftarget

The URI in a Ref-Target header MAY be a relative URI. Similarly, the href in a DAV:reftarget property MAY be a relative URI. In both cases, the base URI to be used for resolving the relative URI to absolute form

is the URI used in the HTTP message to identify the redirect reference to which the Ref-Target entity header or DAV:reftarget property belongs.

In the case of a Ref-Target header, the base URI is constructed as follows: Its scheme component is "http", its authority component is the value of the Host header in the request, and its path component is the request-URI in the request. See Section 5 of [URI] for a discussion of relative URI references and how to resolve them.

The DAV:reftarget property appears in the protocol in the context of a Multi-Status response, in a DAV:response element that contains a single DAV:href element. The value of this DAV:href element serves as the base URI for resolving a relative URI in DAV:reftarget. The value of DAV:href may itself be relative, in which case it must be resolved first in order to serve as the base URI for the relative URI in DAV:reftarget. If the DAV:href element is relative, its base URI is constructed from the scheme component "http", the value of the Host header in the request, and the request-URI.

4.3.9.1 Example: Resolving a Relative URI in Ref-Target

>> Request:

```
MKREF /north/inuvik HTTP/1.1
Host: www.somehost.edu
Ref-Target: mapcollection/inuvik.gif
```

>> Response:

```
HTTP/1.1 201 Created
```

In this example, the base URI is http://www.somehost.edu/north/inuvik. Then, following the rules in [URI] Section 5, the relative URI in Ref-Target resolves to the absolute URI http://www.somehost.edu/north/mapcollection/inuvik.gif.

4.3.9.2 Example: Resolving a Relative URI in DAV:reftarget

>> Request:

```
PROPFIND /geog/ HTTP/1.1
Host: www.xxsvr.com
Passthrough: F
Depth: 1
Content-Type: text/xml
Content-Length: nnn
```

```
<?xml version="1.0" ?>
<D:propfind xmlns:D="DAV:">
   <D:prop>
      <D:resourcetype/>
      <D:reftarget/>
   </D:prop>
</D:propfind>

>> Response:

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

<?xml version="1/0" ?>
<D:multistatus xmlns:D="DAV:">
   <D:response>
      <D:href>/geog/</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><D:reftarget/></D:prop>
         <D:status>HTTP/1.1 404 Not Found</D:status>
     </D:propstat>
   </D:response>
   <D:response>
      <D:href>/geog/stats.html</D:href>
      <D:propstat>
         <D:prop>
            <D:resourcetype><D:redirectref/></D:resourcetype>
            <D:reftarget>statistics/population/1997.html</D:reftarget>
         </D:prop>
         <D:status>HTTP/1.1 200 OK</D:status>
      </D:propstat>
   </D:response>
</D:multistatus>
```

In this example, the relative URI statistics/population/1997.html is returned as the value of reftarget for the reference identified by href /geog/stats.html. The href is itself a relative URI, which resolves to http://www.xxsrv.com/geog/stats.html. This is the base URI for resolving the relative URI in reftarget. The absolute URI of reftarget is http://www.xxsrv.com/geog/statistics/population/1997.html.

4.3.10 Redirect References to Collections

In a Request-URI /segment1/segment2/segment3, any of the three segments

may identify a redirect reference. (See [URI], Section 3.3, for definitions of "path" and "segment".) If any segment in a Request-URI identifies a redirect reference, the response is a 302. The value of the Location header in the 302 response is as follows:

The leftmost path segment of the request-URI that identifies a redirect reference, together with all path segments and separators to the left of it, is replaced by the value of the redirect reference's DAV:reftarget property (resolved to an absolute URI). The remainder of the request-URI is concatenated to this path.

Note: If the DAV:reftarget property ends with a "/" and the remainder of the Request-URI is non-empty (and therefore must begin with a "/"), the final "/" in the DAV:reftarget property is dropped before the remainder of the Request-URI is appended.

Consider Request-URI /x/y/z.html. Suppose that /x/ is a redirect reference whose target is collection /a/, which contains redirect reference y whose target is collection /b/, which contains redirect reference z.html whose target is /c/d.html.

```
/x/ -----> /a/
           /a/y/ -----> /b/
                        /b/z.html -----> /c/d.html
```

In this case the client must follow up three separate 302 responses before finally reaching the target resource. The server responds to the initial request with a 302 with Location: /a/y/z.html, and the client resubmits the request to /a/y/z.html. The server responds to this request with a 302 with Location: /b/z.html, and the client resubmits the request to /b/z.html. The server responds to this request with a **302 with Location: /c/d.html, and the client resubmits the request to** /c/d.html. This final request succeeds.

5 Ordered Collections

5.1 Overview

Collections on a compliant server may be ordered, but need not be. 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 bindings. If a collection is ordered, each of its bindings, and hence internal member URIs, MUST be in the ordering exactly once, and the ordering MUST NOT include any binding that is not contained by the collection. Only one ordering can be attached to any collection. An ordering is considered to be part of the state of a collection resource, and hence is the same across all URI mappings to the 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.

The server is responsible for enforcing these constraints on orderings. The server MUST remove a binding (and its derived internal member URI) from the ordering when it is removed from the collection. The server MUST add a binding (and its derived internal member URI) to the ordering when it is added to the collection.

When responding to a PROPFIND on a collection, the server MUST order the response elements according to the ordering defined on the collection.

If the collection is unordered, the client cannot depend on the repeatability of the ordering of results from a PROPFIND request.

Orderings may be client-maintained or server-maintained. This protocol provides support for both types of orderings.

5.2 Creating an Ordered Collection

5.2.1 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 6.5) 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. This URI may identify a server-maintained ordering. Clients can discover the available server-maintained orderings using the mechanism defined in Section 11.3. The URI may identify a semantics for a client-maintained ordering, providing 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 are discouraged from accessing that resource, in order to avoid overburdening its server. The client MAY set the header value to DAV:custom to indicate that the collection is ordered, but the semantics of the ordering are not being advertised. If the client does not want the collection to be ordered, it may omit the Ordered header, or use it with the value DAV:unordered.

If the server does not recognize the value of the Ordered header as one of its server-maintained orderings, it MUST assume that a client-maintained ordering is intended. If the value of the Ordered header is one of the server-maintained orderings that the server supports, it MUST maintain the collection's ordering according to that ordering semantics

as new members are added.

Every collection MUST have a DAV:orderingtype property (defined in Section 7.5), 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 the collection is unordered, the DAV:orderingtype property MUST have the value 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.

5.2.2 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.

5.3 Setting the Position of a Collection Member

5.3.1 Overview

When a new member is added to a collection with a client-maintained ordering (for example, with PUT, MKREF, or MKCOL), its position in the ordering can be set with the new Position header (defined in Section 6.6). The Position header allows the client to specify that the member should be first in the collection's ordering, last in the collection's ordering, immediately before some other binding in the collection's ordering, or immediately after some other binding in the collection's ordering.

5.3.2 Status Codes

409 (Conflict): The request specifies a position that is before or after a URI that is not an internal member URI of the collection, or before or after itself.

425 (Unordered Collection): The request specifies a collection position in an unordered collection or in a collection with a server-maintained ordering.

5.3.3 Examples: Setting the Position of a Collection Member

>>Request:

```
MKREF /~whitehead/dav/spec08.ref HTTP/1.1
HOST: www.ics.uci.edu
Ref-Target: http://www.ics.uci.edu/i-d/draft-webdav-protocol-08.txt
Position: After <requirements.html>
```

>>Response:

```
HTTP/1.1 201 Created
```

This request resulted in the creation of a new referential resource at www.ics.uci.edu/~whitehead/dav/spec08.ref, which points to the resource identified by the Ref-Target header. The Position header in this example caused the server to set its position in the ordering of the /~whitehead/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/1.1 425 Unordered Collection
```

In this case, the server returned a 425 (Unordered Collection) status code because the /~whitehead/dav/ collection is an unordered collection. Consequently, the server was unable to satisfy the Position header.

5.4 Changing the Semantics of a Collection Ordering

After a collection has been created, a client can change its ordering

semantics, or change an ordered collection to an unordered collection or vice versa, by using PROPPATCH to change the value of its DAV:orderingtype property (defined in Section 7.5). If the new value identifies a client-maintained ordering, the client is then responsible for updating the collection's ordering according to the new semantics. If it identifies a server-maintained ordering, the server MUST reorder the collection according to the new semantics. PROPPATCH is defined in [WebDAV], Section 7.2.

5.5 Changing the Position of a Collection Member

5.5.1 ORDERPATCH Method

The ORDERPATCH method alters the ordering of bindings in the collection identified by the Request-URI, based on instructions in the order XML element. The order XML element identifies the bindings whose positions are to be changed, and describes 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 binding, or immediately after some other binding.

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.

5.5.2 Status Codes

Since multiple changes can be requested in a single ORDERPATCH request, 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): The request specifies a position that is before or after a URI that is not an internal member URI of the collection, or before or after itself.

425 (Unordered Collection): The request specifies a collection position in an unordered collection or in a collection with a server-maintained ordering.

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

5.5.3 Example: Changing Positions in an Ordered Collection

Consider a collection /coll-1/ with bindings 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:href>nunavut.map</d:href>
         </d:after>
      </d:position>
   </d:ordermember>
   <d:ordermember>
      <d:href>iqaluit.img</d:href>
      <d:position>
         <d:last/>
      </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/1.1 200 OK</d:status>
   </d:response>
   <d:response>
      <d:href>http://www.nunanet.com/coll-1/iqaluit.img</d:href>
      <d:status>HTTP/1.1 200 OK</d:status>
   </d:response>
</d:multistatus>
```

If the href elements are relative URIs, as in this example, they are interpreted relative to the collection that is being reordered. In this example, after the request has been processed, the collection's ordering is as follows:

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

5.5.4 Example: Failure of an ORDERPATCH Request

Consider a collection /coll-1/ with bindings 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:href>nunavut.map</d:href>
         </d:after>
      </d:position>
   </d:ordermember>
   <d:ordermember>
      <d:href>iqaluit.map</d:href>
      <d:position>
         <d:after>
            <d:href>pangnirtung.img</d:href>
         </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 binding that is not contained in 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.

6 Headers

6.1 All-Bindings Request Header

```
All-Bindings = "All-Bindings" ":"
```

The All-Bindings request header may be used with DELETE requests to instruct the server to remove all bindings to the resource identified by the Request-URI.

6.2 Ref-Target Entity Header

```
Ref-Target = "Ref-Target" ":" (absoluteURI | relativeURI)
```


The Ref-Target header is defined primarily for use with MKREF requests to identify the target resource of the new redirect reference being created.

6.3 Resource-Type Entity Header

```
Resource-Type = "Resource-Type" ":" ("DAV:redirectref" |
                                     ext-resource-type)
ext-resource-type = coded-URL
```

The Resource-Type header is defined primarily for use in 302 responses, to allow reference-aware clients to distinguish between HTTP 1.1 redirects and 302 responses for redirect references (see Sections 4.1, and 4.3.7). The possible values of this header are DAV:redirectref, and ext-resource-type. The ext-resource-type production is provided for extensibility.

6.4 Passthrough Request Header

```
Passthrough = "Passthrough" ":" ("T" | "F")
```

The optional Passthrough header can be used on any request to a redirect reference. If the Passthrough header has the value "F", the request MUST be applied to the reference itself, and a 302 response MUST NOT be returned. If the Passthrough header has the value "T", a 302 response MUST be returned, with the URI of the target resource in the Location header and the Resource-Type header with a value "DAV:redirectref".

If the Passthrough header is used on a request to any other sort of resource besides a reference, the server SHOULD ignore it. If the Passthrough header with the value "F" appears in a POST or ORDERPATCH request to a reference, the server MUST respond with a 400 (Bad Request).

6.5 Ordered Entity Header

```
Ordered = "Ordered" ":" ("DAV:unordered" | "DAV:custom" | absoluteURI)
```

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 absoluteURI value indicates that the collection is ordered, and identifies the semantics of the ordering.

6.6 Position Request Header

```
Position = "Position" ":" ("First" | "Last" |
                           (("Before" | "After") Generic-Coded-url))
Generic-Coded-url = "<" (absoluteURI | relativeURI) ">"
```

absoluteURI and relativeURI are defined in [URI].

The Position header may be used with any method that adds a binding to a

collection with a client-maintained ordering, to tell the server where in the collection ordering to position the new binding being added to the collection.

If the Generic-Coded-url is a relative URL, it is interpreted relative to the collection to which the new binding is being added.

The server MUST insert the new binding into the ordering at the location specified in the Position header, if one is present (and if the collection has a client-maintained ordering).

The "First" keyword indicates the new binding is put in the beginning position in the collection's ordering, while "Last" indicates the new binding is put in the final position in the collection's ordering. The "Before" keyword indicates the new binding is added to the collection's ordering immediately prior to the position of the binding identified in the Generic-Coded-url. Likewise, the "After" keyword indicates the new binding is added to the collection's ordering immediately following the position of the binding identified in the Generic-Coded-url.

If the request is replacing an existing resource, and the Position header is present, the server MUST remove the binding from its previous position, and then insert it at the requested position.

If the Position request header is not used when adding a binding to a collection with a client-maintained ordering, 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 binding to the collection, the server MUST append the new binding to the end of the ordering.

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

7 Status Codes

7.1 506 Loop Detected

The 506 (Loop Detected) status code indicates that the server detected an infinite loop while processing a request with "Depth: infinity".

7.2 425 Unordered Collection

The 425 (Unordered Collection) status code indicates that the client attempted to set the position of an internal collection member in an unordered collection or in a collection with a server-maintained ordering.

8 Properties

8.1 reftarget Property

Name: reftarget
Namespace: DAV:
Purpose: A property of redirect references that provides an efficient way for clients to discover the URI of the target resource. This is a read-only property, whose value can only be set by using the Ref-Target header with a MKREF request.
Value: URI of the target resource. This value MAY be a relative URI. The reftarget property can occur in the entity bodies of responses to PROPFIND requests.

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

8.2 location Property

Name: location
Namespace: DAV:
Purpose: For use with 302 (Moved Temporarily) response codes in Multi-Status responses. It contains the absolute URI of the temporary location of the resource. In the context of redirect references, this value is the absolute URI of the target resource. It is analogous to the Location header in HTTP 302 responses defined in [HTTP] Section 10.3.3 "302

Moved Temporarily." Including the location property in a Multi-Status response requires an extension to the syntax of the DAV:response element defined in [WebDAV], which is defined in Section 9 below. This property is not expected to be stored on the reference. It is modeled as a property only so that it can be returned inside a DAV:prop element in a Multi-Status response.

Value: href containing the absolute URI of the target resource.

```
<!ELEMENT location href >
```

8.3 bindings Property

Name: bindings
Namespace: DAV:
Purpose: Enables clients to discover, for any resource, what bindings point to it and what collections contain those bindings. This is an optional property. If present, it is a read-only property, maintained by the server.
Value: List of href / segment pairs for all of the bindings that associate URI segments with the resource. The href is an absolute URI for one URI mapping of the collection containing the binding. The segment is the URI segment that identifies the binding within that collection. If a binding belongs to a collection that has multiple URI mappings, only one URI mapping for that collection should be reported.

```
<!ELEMENT bindings ((href, segment)*)>
```

8.4 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.
Value: unordered for an unordered collection, or a URI that uniquely identifies the semantics of the collection's ordering. The value custom indicates that the collection is ordered, but the semantics are not being advertised.

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

9 XML Elements

9.1 redirectref XML Element

Name: redirectref
Namespace: DAV:
Purpose: Used as the value of the DAV:resourcetype property to specify that the resource type is a redirect reference.

```
<!ELEMENT redirectref EMPTY >
```

9.2 segment XML Element

Name: segment
Namespace: DAV:
Purpose: The segment that names a binding, used in the DAV:bindings property.
Value: segment ; as defined in section 3.3 of [URI].

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

9.3 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 >
```

9.4 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 >
```


9.5 order XML Element

Name: order
Namespace: DAV:
Purpose: For use with the new ORDERPATCH method. Describes a change to be made in a collection ordering.

Value: A description of the new positions of the bindings a collection contains in its ordering.

```
<!ELEMENT order (ordermember+) >
```

9.6 ordermember XML Element

Name: ordermember
Namespace: DAV:
Purpose: Occurs in the order XML Element, and describes the new position of a single binding in the collection's ordering.
Value: An href containing a binding'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) >
```

9.7 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 bindings 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 binding, or immediately after some other binding.

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

9.8 first XML Element

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

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

9.9 last XML Element

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

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

9.10 before XML Element

Name: before
Namespace: DAV:
Purpose: Occurs in the position XML element. Specifies that the binding should be placed immediately before the binding in the enclosed href XML element in the collection's ordering.
Value: href of the member it precedes in the ordering

```
<!ELEMENT before href >
```

9.11 after XML Element

Name: after
Namespace: DAV:
Purpose: Occurs in the position XML element. Specifies that the binding should be placed immediately after the binding in the enclosed href XML element in the collection's ordering.
Value: href of the member it follows in the ordering

```
<!ELEMENT after href >
```

9.12 options XML Element

Name: options
Namespace: DAV:
Purpose: Used in OPTIONS requests to ask for more detailed information about capabilities than can be provided in the DAV: response header. Used in OPTIONS responses to provide that information.
Value: List of elements identifying or providing the additional information desired.

```
<!ELEMENT options (orderingoptions | ANY)+ >
```

9.13 orderingoptions XML Element

Name: orderingoptions
Namespace: DAV:
Purpose: Used in OPTIONS requests to ask for the list of server-maintained orderings that can be supported at the request-URI. Used in OPTIONS responses to provide that information. These values can be used in the Ordered header or the DAV:orderingtype property to request that a particular server-maintained ordering be applied to the collection.
Value: EMPTY on requests. On responses, it is the list of server-maintained orderings available for the request-URI.

```
<!ELEMENT orderingoptions ( (#PCDATA)+ | EMPTY) >
```

10 Extensions to the DAV:response XML Element for Multi-Status Responses

As described in Sections 4.6 and 4.9, the DAV:location property and the DAV:reftype property may be returned in the DAV:response element of a **207 Multi-Status response, to allow clients to resubmit their requests**

to the target resource of a redirect reference.

Whenever these properties are included in a Multi-Status response, they are placed in a DAV:prop element associated with the href to which they apply. This structure provides a framework for future extensions by other standards that may need to include additional properties in their responses.

Consequently, the definition of the DAV:response XML element changes to the following:

<!ELEMENT response (href, ((href*, status, prop?) | (propstat+)), responsedescription?) >

11 Capability Discovery

11.1 Compliance Classes

This specification defines OPTIONAL extensions to [WebDAV]. Since resource sharing and ordering are independent capabilities, a resource MAY support either, both, or neither of these capabilities. A resource that provides resource sharing MUST support both bindings and redirect references. A response to an OPTIONS request MUST indicate which of these capabilities the resource supports.

This specification defines three new methods: BIND and MKREF in support of shared resources, and ORDERPATCH in support of ordering. The response MUST indicate which of these methods the resource allows. In addition, the response MUST include the DAV header, as described in Sections 9.1 and 15 of [WebDAV]. Two new compliance classes are defined here for use with the DAV header: sharing and orderedcoll.

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 bindings can be ordered. It implies nothing about whether any collections identified by its internal member URIs can be ordered.

When responding to an OPTIONS request, any type of resource can include sharing in the value of the DAV header. Including sharing indicates that the server permits a redirect reference or a binding at the request URI.

11.2 Example: Discovery of Compliance Classes

```
>> 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, BIND, ORDERPATCH
Public: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, COPY, MOVE, MKCOL,
PROPFIND, PROPPATCH, LOCK, UNLOCK, BIND, MKREF, ORDERPATCH
DAV: 1, 2, sharing, 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 and resource sharing. The Allow header indicates that BIND and ORDERPATCH requests can be submitted to /somecollection/. Since a redirect reference is not a collection, a MKREF request with /somecollection/ as its Request-URI would fail, but the Public header shows that other Request-URIs on the server do support MKREF.

11.3 Additional Advanced Collections Capabilities

Clients may need detailed information about specific areas of advanced collections functionality. This information can be requested by sending an OPTIONS request with an XML body that includes a DAV:options element. The DAV:options element contains a list of empty elements identifying the information the client needs.

As described in Section 5.2, servers may offer a set of server-maintained orderings on collections. Clients can discover the list of server-maintained orderings available for the request-URI by including an empty DAV:orderingoptions element in the DAV:options element. The response will include a DAV:orderingoptions element with the list of supported server-maintained orderings. Servers SHOULD advertise the server-maintained orderings available using this mechanism.

11.4 Example: Discovery of Ordering Options

```
>> Request:

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

<?xml version="1.0" ?>
<D:options xmlns:D="DAV:">
  <D:orderingoptions/>
</D:options>

>> 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, BIND, ORDERPATCH
Public: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, COPY, MOVE, MKCOL,
PROPFIND, PROPPATCH, LOCK, UNLOCK, BIND, MKREF, ORDERPATCH
DAV: 1, sharing, orderedcoll
```



```
<?xml version="1.0" ?>
<D:options xmlns:D="DAV:">
  <D:orderingoptions xmlns:X="Xerox:">
      <X:author-ascending/>
      <X:title-ascending/>
      <X:date-descending/>
  </D:orderingoptions>
</D:options>
```

This response indicates that the resource /somecollection/ is level 1 compliant, as defined in [WebDAV]. In addition, /somecollection/ supports ordering and resource sharing. The client also asked for a list of the server-maintained orderings that are supported for /somecollection/. The response indicates that the orderings Xerox:author-ascending, Xerox:title-ascending, and Xerox:date-descending are supported.

12 Security Considerations

This section is provided to detail issues concerning security implications of which WebDAV applications need to be aware.

All of the security considerations of HTTP/1.1 and the WebDAV Distributed Authoring Protocol specification also apply to WebDAV collections. In addition, resource sharing and ordered collections

introduce several new security concerns and increase the risk of some existing threats. These issues are detailed below.

12.1 Privacy Concerns

By creating redirect references on a trusted server, it is possible for a hostile agent to induce users to send private information to a target on a different server. This risk is mitigated somewhat, since clients are required to notify the user of the redirection for any request other than GET or HEAD. (See [HTTP], Section 10.3.3 Moved Temporarily.)

The same risk exists for bindings, although it is less likely that servers will support cross-server bindings.

12.2 Redirect Loops

Although redirect loops were already possible in HTTP 1.1, the introduction of the BIND and MKREF methods creates a new avenue for clients to create loops accidentally or maliciously. If the binding or reference and its target are on the same server, the server may be able to detect MKREF and BIND requests that would create loops. See also [HTTP], Section 10.3 "Redirection 3xx." Servers are required to detect loops caused by bindings to collections during the processing of any requests with "Depth: infinity".

12.3 Redirect References, Bindings, and Denial of Service

Denial of service attacks were already possible by posting URLs that were intended for limited use at heavily used Web sites. The

introduction of BIND and MKREF creates a new avenue for similar denial of service attacks. Clients can now create bindings and redirect references at heavily used sites to target locations that were not designed for heavy usage.

12.4 Private Locations May Be Revealed

There are several ways that redirect references may reveal information about directory structures. First, the DAV:reftarget property of every redirect reference contains the URI of the target resource. Anyone who has access to the reference can discover the directory path that leads to the target resource. The owner of the target resource may have wanted to limit knowledge of this directory structure.

Sufficiently powerful access control mechanisms can control this risk to some extent. Property-level access control could prevent users from examining the DAV:reftarget property. (The Ref-Target and Location headers, which are returned in some responses to requests on redirect

references, reveal the same information, however.) In some environments, the owner of a resource might be able to use access control to prevent others from creating references to that resource.

In addition, if backpointers are maintained on the target resource, the owners of bindings face these same risks. The directory structures where bindings are located are revealed to anyone who has access to the DAV:bindings property on a target resource. Moving a binding may reveal its new location to anyone with access to DAV:bindings on its target resource.

12.5 DAV:bindings and Denial of Service

If the server maintains the DAV:bindings property in response to bindings created in other administrative domains, it is exposed to hostile attempts to make it devote resources to adding bindings to the list.

12.6 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. In addition, Section 5.2 discourages clients from looking up the semantics at that location.

13 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 [Alvestrand].

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 [Alvestrand].

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.

For rationales for these decisions and advice for application implementors, see [WebDAV].

14 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.

15 Copyright

To be supplied by the RFC Editor.

16 Intellectual Property

To be supplied by the RFC Editor.

17 Acknowledgements

This draft has benefited from thoughtful discussion by Jim Amsden, Steve Carter, 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, Steve Martin, Larry Masinter, Jeff McAffer, Surendra Koduru Reddy, Max Rible, Sam Ruby, Bradley Sergeant, Nick Shelness, John Stracke, John Tigue, John Turner, and others.

18 References

18.1 Normative References

[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, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1." RFC 2068. UC Irvine, DEC, MIT/LCS. January, 1997.

[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.

18.2 Informational References

[DASL] Saveen Reddy, D. Jensen, Surendra Reddy, R. Henderson, J. Davis, A. Babich, "DAV Searching & Locating." Draft-reddy-dasl-protocol-03. Internet Draft, work in progress. Microsoft, Novell, Oracle, Netscape, Xerox, Filenet. November, 1998.

[CollReq] J. Slein, J. Davis, "Requirements for Advanced Collection Functionality in WebDAV." Draft-ietf-webdav-collection-reqts-02. Internet Draft, work in progress. Xerox. February, 1999.

19 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
Email: ccjason@us.ibm.com

T. Chihaya
DataChannel, Inc.
155 108th Ave. N.E., Suite 400
Bellevue, WA 98004
Email: Tyson@DataChannel.com

20 Appendices

20.1 Appendix 1: Extensions to the WebDAV Document Type Definition

```
<!--============= XML Elements from Section 8 ================-->
<!ELEMENT redirectref EMPTY >
<!ELEMENT segment (#PCDATA)>
<!ELEMENT unordered EMPTY >
<!ELEMENT custom EMPTY >
<!ELEMENT order (ordermember+) >
<!ELEMENT ordermember (href, position) >
<!ELEMENT position (first | last | before | after)>
<!ELEMENT first EMPTY >
<!ELEMENT last EMPTY >
<!ELEMENT before href >
<!ELEMENT after href >
<!ELEMENT options (refintegrityoptions | orderingoptions)+ >
<!ELEMENT orderingoptions ( (#PCDATA)+ | EMPTY) >
<!--============= Property Elements from Section 7 ==================-->
<!ELEMENT reftarget (#PCDATA)>
<!ELEMENT location href>
<!ELEMENT bindings ((href, segment)*)>
<!ELEMENT orderingtype (arbitrary | custom | href) >
<!--====== Changes to the DAV:response Element from Section 9 ====-->
<!ELEMENT response (href, ((href*, status, prop?) | (propstat+)),
responsedescription?) >
```