WEBDAV Working Group
INTERNET DRAFT
<draft-ietf-webdav-binding-protocol-02.txt>

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

December 17, 1999

Expires June 17, 2000

WebDAV Bindings

Status of this Memo

This document is an Internet-Draft and is in full conformance with all provisions of Section 10 of RFC2026. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress".

To view the list Internet-Draft Shadow Directories, see http://www.ietf.org/shadow.html.

Distribution of this document is unlimited. Please send comments to the Distributed Authoring and Versioning (WebDAV) working group at <w3c-dist-auth@w3.org>, which may be joined by sending a message with subject "subscribe" to <w3c-dist-auth-request@w3.org>.

Discussions of the WEBDAV working group are archived at URL: <http://lists.w3.org/Archives/Public/w3c-dist-auth/>.

Abstract

This is one of a pair of specifications that extend the WebDAV Distributed Authoring Protocol to enable clients to create new access paths to existing resources. The two protocol extensions have very different characteristics that make them useful for different sorts of applications.

The present specification defines bindings, and the BIND method for creating them. Creating a new binding to a resource indirectly creates one or more new URIs mapped to that resource, which can then be used to access it. Servers are required to insure the integrity of any bindings that they allow to be created.

The related specification, RFC xxxx, defines redirect reference resources. A redirect reference resource is a resource whose default response is an HTTP/1.1 302 (Found) status code, redirecting the client

to a different resource, the target resource. A redirect reference makes it possible to access the target resource indirectly, through any URI mapped to the redirect reference resource. There are no integrity guarantees associated with redirect reference resources.

Table of Contents

1 Notational Conventions

Since this document describes a set of extensions to the WebDAV Distributed Authoring Protocol [WebDAV], itself an extension to the HTTP/1.1 protocol, the augmented BNF used here to describe protocol elements is exactly the same as described in Section 2.1 of [HTTP].

Since this augmented BNF uses the basic production rules provided in Section 2.2 of [HTTP], these rules apply to this document as well.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2 Introduction

This is one of a pair of specifications that extend the WebDAV Distributed Authoring Protocol to enable clients to create new access paths to existing resources. This capability is useful for several reasons:

URIs of WebDAV-compliant resources are hierarchical and correspond to a hierarchy of collections in resource space. The WebDAV Distributed Authoring Protocol makes it possible to organize these resources into hierarchies, placing them into groupings, known as collections, which are more easily browsed and manipulated than a single flat collection. 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. Allowing clients to create new URIs that access the existing resource lets them put that resource into multiple collections.

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. Being able to create new access paths to existing resources in other collections or even on other servers is useful for this sort of case.

The BIND method defined here provides a mechanism for allowing clients to create alternative access paths to existing WebDAV resources. HTTP and WebDAV methods are able to work because there are mappings between URIs and resources. A method is addressed to a URI, and the server follows the mapping from that URI to a resource, applying the method to that resource. Multiple URIs may be mapped to the same resource, but until now there has been no way for clients to create additional URIs mapped to existing resources.

BIND lets clients associate a new URI with an existing WebDAV resource, and this URI can then be used to submit requests to the resource. Since URIs of WebDAV resources are hierarchical, and correspond to a hierarchy of collections in resource space, the BIND method also has the effect of adding the resource to a collection. As new URIs are associated with the resource, it appears in additional collections.

The companion specification, RFC xxxx, defines redirect reference resources, a different mechanism for creating alternative access paths to existing resources. A redirect reference is a resource in one collection whose purpose is to forward requests to another resource (its target), usually in a different collection. In this way, it provides access to the target resource from another collection. It redirects most requests to the target resource using the HTTP 302 (Moved Temporarily) status code, thereby providing a form of mediated access to the target resource.

Bindings and redirect reference resources have very different characteristics:

A BIND request does not create a new resource, but simply makes available a new URI for submitting requests to an existing resource. The new URI is indistinguishable from any other URI when submitting a request to a resource. Only one round trip is needed to submit a request to the intended target. Servers are required to enforce the integrity of the relationships between the new URIs and the resources associated with them. Consequently, it may be very costly for servers to support BIND requests that cross server boundaries.

A redirect reference is a resource, and so can have properties of its own. Properties of the redirect reference can contain such information as who created the reference, when, and why. Since redirect references are implemented using HTTP 302 responses, it generally takes two round

trips to submit a request to the intended resource. Servers are not required to enforce the integrity of redirect references. Redirect references work equally well for local resources and for resources that reside on a different server from the reference.

The remainder of this specification is organized as follows. Section 3 defines terminology used in the rest of the specification, while Section **4 briefly overviews bindings.** Section 5 specifies the BIND method, used to create bindings. Sections 6 through 9 discuss the relationships between bindings and other HTTP and WebDAV methods. Sections 10 and 11 define mechanisms for tracking bindings. Sections 12 through 14 define the new status codes, properties, and XML elements needed to support bindings. Section 15 discusses compliance and capability discovery. Security concerns, internationalization issues, and IANA considerations are described in Sections 16 through 18. The remaining sections provide other supporting information.

3 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].

URI Mapping
: A relation between an absolute URI and a resource. For an absolute URI U and the resource it identifies R, the URI mapping can be thought of as (U => R). 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. 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 URI. Formally, as defined in section 3.3 of [URI].

Binding
: A relation between a single path segment (in a collection) and a resource. A binding is part of the state of a collection. If two different collections contain a binding between the same path segment and the same resource, these are two distinct bindings. So for a collection C, a path segment S, and a resource R, the binding can be thought of as C:(S -> R). Bindings create URI mappings, and hence allow requests to be sent to a single resource from multiple locations in a URI namespace. For example, given

o collection C, accessible through the URI
  http://www.srv.com/coll/,
o path segment S, equal to "foo.html", and
o resource R,

creating the binding C: (S -> R) makes it possible to use the URI http://www.srv.com/coll/foo.html to access R.

The following figure illustrates a more complex example of how bindings create URI mappings.

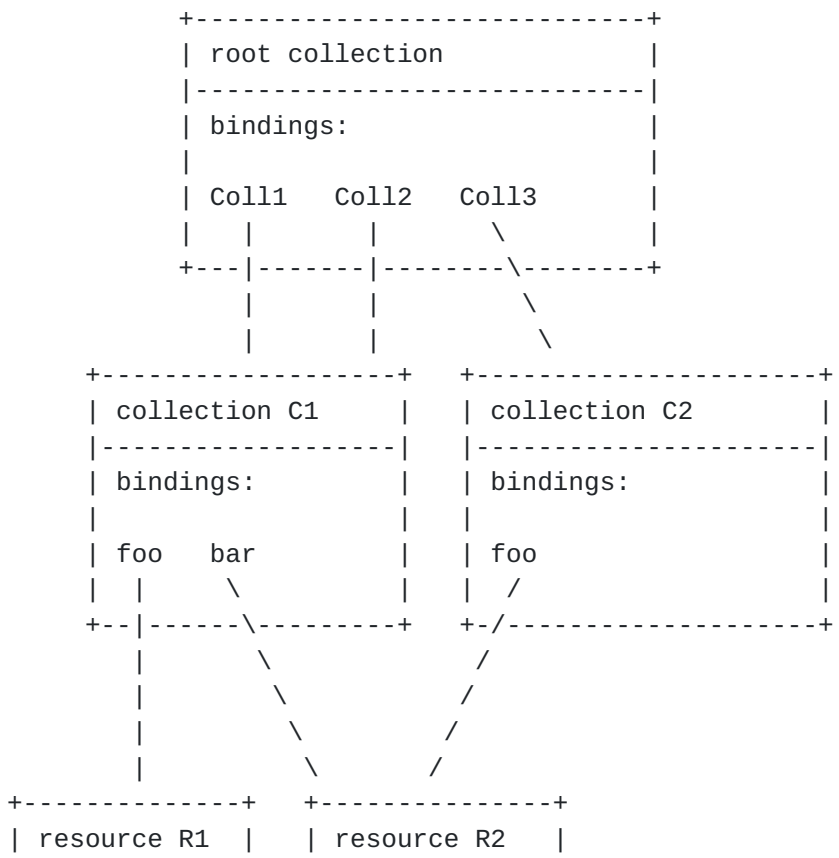


Figure 1

Since there are two bindings in the root collection, Coll1 and Coll2, to collection C1, the single binding C1:(foo -> R1) between foo and resource R1 in collection C1 creates two URI mappings, /Coll1/foo and /Coll2/foo, to resource R1. Each of these URI mappings can be used to submit requests to R1. The binding C1:(bar -> R2) between bar and resource R2 in collection C1 and the binding C2:(foo -> R2) between foo and resource R2 in

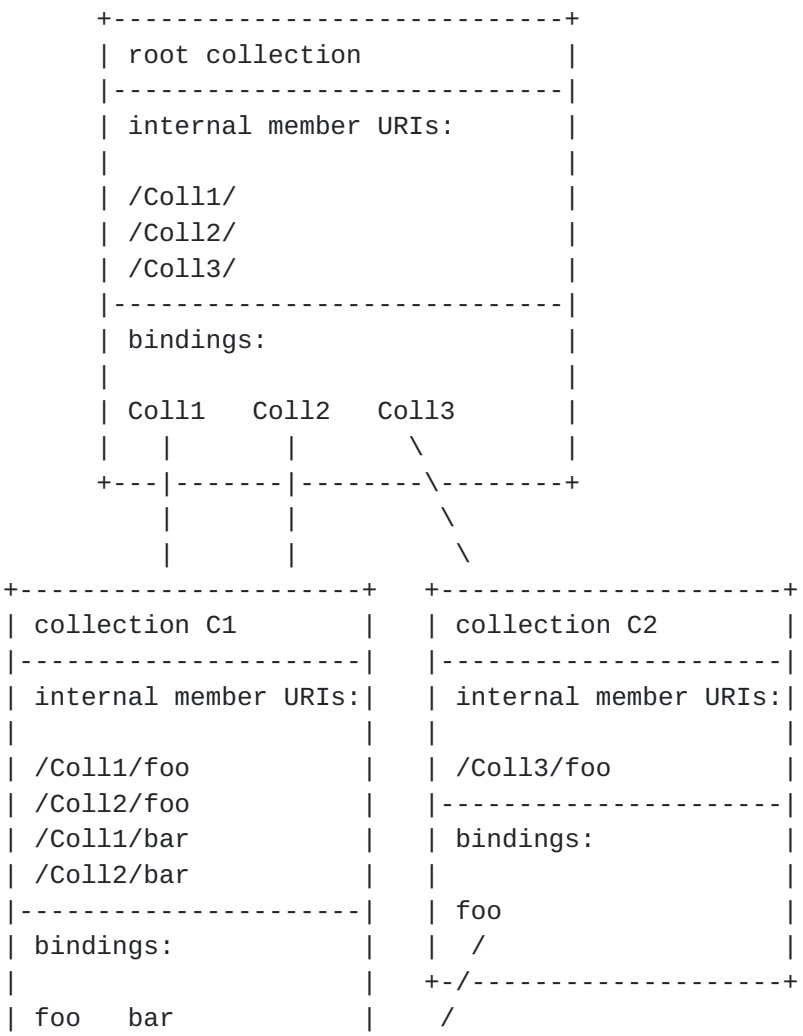
collection C2 create altogether 3 URI mappings to resource R2: /Coll1/bar, /Coll2/bar, and /Coll3/foo. All 3 URI mappings can be used to submit requests to resource R2.

Collection

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

Internal Member URI

Informally, the complete set of URLs by which a collection member is known. Formally, the URI U of a URI mapping (U => R), created by a binding that is contained in a collection. The following figure illustrates the relationship between bindings and internal member URIs in a collection:



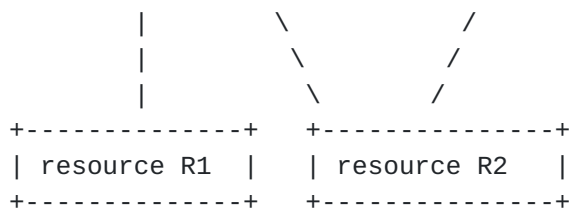


Figure 2

The URIs of all URI mappings created by a collection's bindings are internal member URIs of the collection.

However, for a given request, only the URIs from those URI mappings that incorporate the Request-URI are treated as internal member URIs. This is done to prevent large amounts of duplicate information from being returned for operations on collections. The problem would occur if a PROPFIND on a collection to which there are multiple URI mappings returned information for every URI mapping.

For example, in Figure 2 above, if a PROPFIND request with "Depth: infinity" is submitted to collection C1 using the Request-URI /Coll1/, only the URI mappings starting with the Request-URI would be listed as internal member URIs. The response would include only /Coll1/ itself and the internal member URIs /Coll1/foo and /Coll1/bar. It would not include /Coll2/, /Coll2/foo, or /Coll2/bar, even though the URI /Coll2/ does map to the collection, and /Coll2/foo and /Coll2/bar are internal member URIs of the collection.

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 URI 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 the URI of a URI 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.

3.1 Rationale for Distinguishing Bindings from URI Mappings

Consider again collection C1 in Figure 2. If we had only the notion of URI mappings, we would be forced to say that C1's membership was defined

by the list of internal member URIs. If these URIs identify the

membership, and are part of the state of the collection, then the act of making the collection available via a new URI has the effect of changing the collection's membership, hence changing the collection's state. This is undesirable, since ideally a collection's membership should remain the same, if suddenly a new URI can be used to access the collection. What is needed is a way to separate the final segment of a URI from the collection's URI contribution.

The notion of binding is introduced to separate the final segment of a URI from its parent collectionÆs contribution. This done, a collection can be defined as containing a set of bindings, thus permitting new mappings to a collection without modifying its membership. We introduce the concept of URI mapping to combine together the collection's URI and a binding's segment to create a full URI that can be used in protocol requests. Finally, the internal member URI, first defined in [WebDAV], is redefined here to maintain backward compatibility with that specification.

4 Overview of Bindings

Bindings are part of the state of a collection. In general, there is a one-to-many correspondence between a collection's bindings and its internal member URIs, as illustrated in Figure 2 above. The URI segment associated with a resource by one of a collection's bindings is also the final segment of one or more 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.

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.

The identity of a binding C:(S -> R) is determined by the URI segment (in its collection) and the resource that the binding associates. If the resource goes out of existence (as a result of some out-of-band operation), 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.

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 moving a resource through one binding to disrupt another binding, turning that binding into a dangling path segment. Nor is it acceptable for a server, after removing one binding, to reclaim

the system resources associated with its resource while other bindings to the resource remain. Implementations MUST ensure the integrity of bindings.

5 BIND Method

5.1 Overview of BIND

The BIND method creates a new binding between the resource identified by

the Request-URI and the final segment of the Destination header (minus any trailing slash). This binding is added to the collection identified by the Destination header 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 here, including the guarantee of the integrity of the binding, the BIND request MUST fail.

Note: 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 another server that manages a binding to the resource. 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. Status code 507 (Cross-server Binding Forbidden) is defined in Section **12.2 for cases where servers must fail cross-server BIND requests** because they cannot guarantee the integrity of cross-server bindings.

If the Destination header does not contain a path segment (i.e., it consists simply of a slash "/"), the BIND operation MUST fail and report a 400 (Bad Request) status code. A binding consists of a (collection, segment, resource) triple, and the Destination header is required to specify the collection and segment of this triple.

If the Destination header minus its final path segment does not identify a collection, the BIND operation MUST fail and report a 400 (Bad Request) status code.

Note: Section 5.1 of [WebDAV] defines a consistent namespace to be one such that "for every URL in the HTTP hierarchy there exists a collection that contains that URL as an internal member," but [WebDAV] does not require namespaces to be consistent. There can exist URIs that map to resources, but do not depend on the existence of collections for the

mapping. For example, the URI http://www.svr.com/x/y.html may map to resource R even if the URI http://www.svr.com/x/ does not map to any resource. This specification does not support the creation of such URI mappings. The BIND method can be used only in consistent sections of a namespace.

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

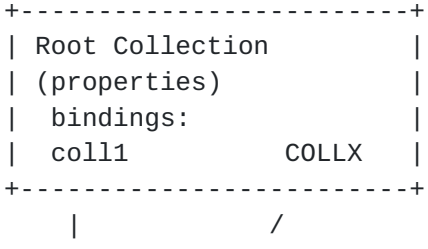
By default, if the Destination header identifies an existing binding, the new binding replaces the existing binding. This default binding replacement behavior can be overridden using the Overwrite header defined in Section 9.6 of [WebDAV].

5.2 Bindings to Collections

Bindings to collections can result in loops. If a server wants to prevent a loop from being created, it MAY fail the BIND request with a **403 (Forbidden) status code.** If a server allows a loop to be created, it MUST detect the loop when processing "Depth: infinity" requests that encounter the loop. It is sometimes possible to complete an operation in spite of the presence of a loop. However, the 506 (Loop Detected) status code is defined in Section 12.1 for use in contexts where an operation is terminated because a loop was encountered.

Creating a new binding to a collection makes each resource associated with a binding in that collection accessible via a new URI, and thus creates new URI mappings to those resources but no new bindings.

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.



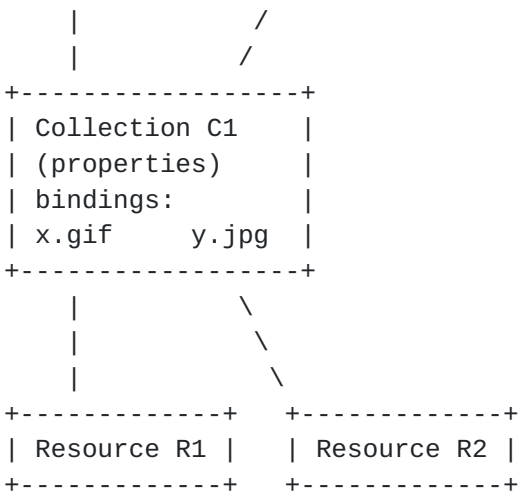


Figure 3

5.3 URI Mappings Created by a BIND

Suppose a BIND request causes a binding from "Binding-Name" to resource R to be added to a collection, C. Then if C-MAP is the set of URI's that were mapped to C before the BIND request, then for each URI "C-URI" in C-MAP, the URI "C-URI/Binding-Name" is mapped to resource R following the BIND request.

Note that if R is a collection, additional URI mappings are created to the descendents of R. Also note that if a binding is made in collection C to C itself (or to a parent of C), an infinite number of mappings is introduced.

5.4 Example: URI Mappings Created by a BIND

For example, if a binding from "foo.html" to R is added to a collection C, and if the following URI's are mapped to C:

http://www.fuzz.com/A/1
http://fuzz.com/A/one

then the following new mappings to R are introduced:

http://www.fuzz.com/A/1/foo.html
http://fuzz.com/A/one/foo.html

If a binding from "myself" to C is then added to C, the following infinite number of additional mappings to C are introduced:

http://www.fuzz.com/A/1/myself
http://www.fuzz.com/A/1/myself/myself

...

and the following infinite number of additional mappings to R are introduced:

http://www.fuzz.com/A/1/myself/foo.html
http://www.fuzz.com/A/1/myself/myself/foo.html
...

5.5 BIND Status Codes

201 (Created): The binding was successfully created.

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

403 (Forbidden): This server has a policy that forbids creation of bindings that would result in loops.

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

507 (Cross-Server Binding Forbidden): The server is unable to create the requested binding because it would bind a segment in a collection on one server to a resource on a different server.

5.6 Example: BIND

>> Request:

```
BIND /pub/i-d/draft-webdav-protocol-08.txt HTTP/1.1
```



```
Host: www.ics.uci.edu
Destination: http://www.ics.uci.edu/~whitehead/dav/spec08.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 URI in the Destination header, "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/.

6 DELETE and Bindings

The DELETE method was originally defined in [HTTP]. This section

redefines the behavior of DELETE in terms of bindings, an abstraction not available when writing [HTTP]. [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. The definition presented here is a clarification of the definition in [HTTP].

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.

Once a resource is unreachable by any URI mapping, the server MAY reclaim system resources associated with that resource. If DELETE removes a binding to a resource, but there remain URI mappings to that resource, the server MUST NOT reclaim system resources associated with the resource.

Although [WebDAV] allows a DELETE to be a non-atomic operation, the DELETE operation defined here is atomic. In particular, a DELETE on a hierarchy of resources is simply the removal of a binding to the collection identified by the Request-URI, and so is a single (and therefore atomic) operation.

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 MUST NOT follow this requirement.

7 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, 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.

Figure 4 below shows an example: Suppose that a COPY 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. 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).

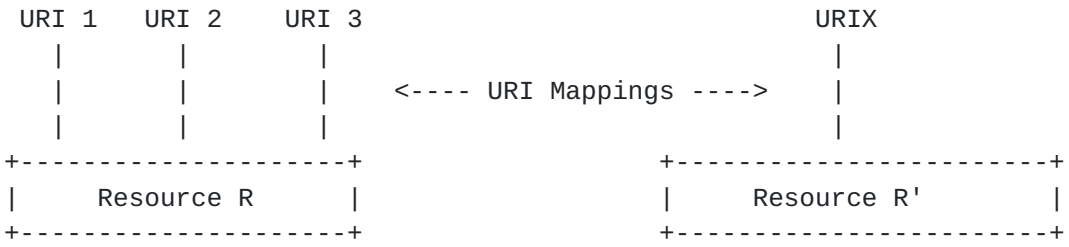


Figure 4

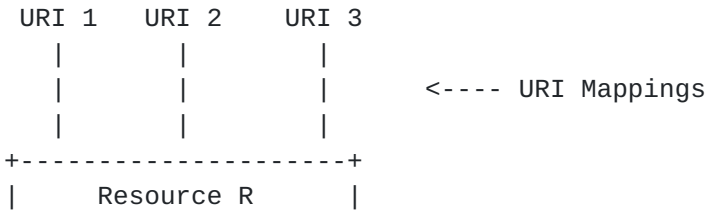
It might be thought that a COPY request with "Depth: 0" on a collection would duplicate its bindings, since bindings are part of the collection's state. This is not the case, however. The definition of Depth in [WebDAV] makes it clear that a "Depth: 0" request does not apply to a collection's members. Consequently, a COPY with "Depth: 0" does not duplicate the bindings contained by the collection.

8 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:

 URI 1   URI 2    URIX
   |       |        |
   |       |        |      <---- URI Mappings
   |       |        |
+----------------------+
|     Resource R       |
+----------------------+
```

Figure 5

Although [WebDAV] allows a MOVE on a collection to be a non-atomic operation, the MOVE operation defined here MUST be atomic. Even when the Request-URI identifies a collection, the MOVE operation involves only removing one binding to that collection and adding another. There are no operations on bindings to any of its children, so the case of MOVE on a collection is the same as the case of MOVE on a non-collection resource. Both are atomic.

9 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 these methods, no new complexities are introduced by allowing explicit creation of multiple bindings to the same resource. When applied to static resources (that is, resources that are not CGI scripts, Active Server Pages, etc.), they obey the following rule:

o A method submitted through one URI mapping, on success, MUST produce the same results as the same method, with the same headers and entity body, submitted through any other URI mapping to the same resource.

When applied to dynamic resources, it is not possible to state any such rule. For any method, a dynamic resource may be sensitive to the URI mapping used to access it. The resource may produce different results depending upon which URI mapping was used to submit the request.

Nevertheless, servers MAY allow new bindings to dynamic resources to be created using BIND.

10 Determining Whether Two Bindings Are to the Same Resource

It is useful to have some way of determining whether two bindings are to the same resource. Two different resources might have identical contents and identical values for the properties defined in [WebDAV]. Although the DAV:bindings property defined in Section 13.1 provides this information, it is an optional property.

The REQUIRED DAV:resourceid property defined in Section 13.2 is a

resource identifier, which MUST be unique across all resources for all time. If the values of DAV:resourceid returned by PROPFIND requests through two bindings are identical, the client can be assured that the two bindings are to the same resource.

The DAV:resourceid property is created, and its value assigned, when the resource is created. The value of DAV:resourceid MUST NOT be changed. Even after the resource is no longer accessible through any URI, that value MUST NOT be reassigned to another resource's DAV:resourceid property.

Any method that creates a new resource MUST assign a new, unique value to its DAV:resourceid property. For example, a PUT that creates a new resource must assign a new, unique value to its DAV:resourceid property. A COPY, since it creates a new resource at the Destination URI, must assign a new, unique value to its DAV:resourceid property.

On the other hand, any method that affects an existing resource MUST NOT change the value of its DAV:resourceid property. For example, a PUT that updates an existing resource must not change the value of its DAV:resourceid property. A MOVE, since it does not create a new resource, but only changes the location of an existing resource, must not change the value of its DAV:resourceid property.

10.1 resourceid URI Scheme

The value of DAV:resourceid is a URI and may use any URI scheme that guarantees the uniqueness of the value across all resources for all time. The resourceid URI scheme defined here is an example of an acceptable URI scheme.

The resourceid URI scheme requires the use of the Universal Unique Identifier (UUID) mechanism, as described in [ISO-11578]. UUID generators may choose between two methods of creating the identifiers. They can either generate a new UUID for every identifier they create or they can create a single UUID and then add extension characters. If the second method is selected, then the program generating the extensions MUST guarantee that the same extension will never be used twice with the associated UUID.

resourceid-URI = "resourceid:" UUID [Extension] ; The UUID production is the string representation of a UUID, as defined in [ISO-11578]. Note that white space (LWS) is not allowed between elements of this production.

Extension = path ; path is defined in Section 3.3 of [URI].

11 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. If the DAV:bindings property exists on a given resource, it MUST contain a complete list of all bindings to that resource. A PROPFIND requesting DAV:bindings MUST return only those bindings that the client is authorized to see. By retrieving this property, a client can discover

the bindings that point to the resource and the collections that contain bindings to the resource.

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 16.4 and 16.5.

12 Status Codes

12.1 506 Loop Detected

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

When this status code is the top-level status code for the operation, it indicates that the entire operation failed.

When this status code occurs inside a multistatus response, it indicates only that a loop is being terminated, but does not indicate failure of the operation as a whole.

For example, consider a PROPFIND request on the following collection:

```
Coll-1 (bound to collection C)
        Foo (bound to resource R)
        Bar (bound to collection C)

>> Request:

PROPFIND /Coll-1/ HTTP/1.1
```

```
Host: www.somehost.com
Depth: infinity
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx

<?xml version="1.0" encoding="utf-8" ?>
<D:propfind xmlns:D="DAV:">
   <D:prop>
      <D:displayname/>
   </D:prop>
</D:propfind>

>> Response:

HTTP/1.1 207 Multi-Status
```



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

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:">
   <D:response>
      <D:href>http://www.somehost.com/Coll-1/</D:href>
      <D:propstat>
         <D:prop>
            <D:displayname>Loop Demo</D:displayname>
         </D:prop>
         <D:status>HTTP/1.1 200 OK</D:status>
      </D:propstat>
   </D:response>
   <D:response>
      <D:href>http://www.somehost.com/Coll-1/Foo</D:href>
      <D:propstat>
         <D:prop>
            <D:displayname>Bird Inventory</D:displayname>
         </D:prop>
         <D:status>HTTP/1.1 200 OK</D:status>
      </D:propstat>
   </D:response>
   <D:response>
      <D:href>http://www.somehost.com/Coll-1/Bar</D:href>
      <D:propstat>
         <D:prop>
            <D:displayname/>
         </D:prop>
         <D:status>HTTP/1.1 506 Loop Detected</D:status>
      </D:propstat>
   </D:response>
```

```
</D:multistatus>
```

12.2 507 Cross-Server Binding Forbidden

The server is unable to create the requested binding because it would bind a segment in a collection on one server to a resource on a different server.

13 Properties

13.1 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 on a given
            resource, it is a read-only property, maintained by the
            server, and contains a complete list of all bindings to that
            resource.
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.  Since there may be multiple URI
            mappings for this collection, it is necessary to select one
            (preferably the URI mapping contained in the Request-URI of
            the BIND request) for use in the DAV:bindings property. The
            segment is the URI segment that identifies the binding
            within that collection.

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

13.2 resourceid Property

```
Name:       resourceid
Namespace:  DAV:
Purpose:    Enables clients to determine whether two bindings are for
            the same resource.
Value:      URI that is guaranteed unique across all resources for all
            time.  It may be of the resourceid URI scheme
            defined in Section 10.1 or any other URI scheme that
            guarantees this uniqueness.

<!ELEMENT resourceid (href)>
```

14 XML Elements

14.1 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)>
```

15 Capability Discovery

Sections 9.1 and 15 of [WebDAV] describe the use of compliance classes with the DAV header in responses to OPTIONS, to indicate which parts of the WebDAV Distributed Authoring protocol the resource supports. This specification defines an OPTIONAL extension to [WebDAV]. It defines a new compliance class, called bindings, for use with the DAV header in responses to OPTIONS requests. If a resource does support bindings, its response to an OPTIONS request may indicate that it does, by listing the new BIND method as one it supports, and by listing the new bindings compliance class in the DAV header.

When responding to an OPTIONS request, any type of resource can include bindings in the value of the DAV header. Doing so indicates that the server permits a binding at the request URI.

15.1 Example: Discovery of Support for Bindings

```
>> Request:

OPTIONS /somecollection/someresource 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
Public: OPTIONS, GET, HEAD, POST, PUT, DELETE, TRACE, COPY, MOVE, MKCOL,
PROPFIND, PROPPATCH, LOCK, UNLOCK, BIND, MKREF, ORDERPATCH
DAV: 1, 2, bindings
```

The DAV header in the response indicates that the resource /somecollection/someresource is level 1 and level 2 compliant, as defined in [WebDAV]. In addition, /somecollection/someresource supports bindings. The Allow header indicates that BIND requests can be

submitted to /somecollection/someresource. The Public header shows that other Request-URIs on the server support additional methods.

16 Security Considerations

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

All of the security considerations of HTTP/1.1 and the WebDAV Distributed Authoring Protocol specification also apply to this protocol specification. In addition, bindings introduce several new security concerns and increase the risk of some existing threats. These issues are detailed below.

16.1 Privacy Concerns

In a context where cross-server bindings are supported, creating bindings on a trusted server may make it possible for a hostile agent to induce users to send private information to a target on a different server.

16.2 Redirect Loops

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

16.3 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 creates a new avenue for similar denial of service attacks. If cross-server bindings are supported, clients can now create

bindings at heavily used sites to target locations that were not designed for heavy usage.

16.4 Private Locations May Be Revealed

If the DAV:bindings property is maintained on a resource, the owners of the bindings risk revealing private locations. The directory structures where bindings are located are available to anyone who has access to the DAV:bindings property on the resource. Moving a binding may reveal its new location to anyone with access to DAV:bindings on its resource.

16.5 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.

17 Internationalization Considerations

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

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

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

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

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

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

18 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.

In addition, this document defines the new resourceid URI Scheme in Section 10.1 and the new HTTP/1.1 status codes 506 and 507in Section 12.

19 Copyright

To be supplied by the RFC Editor.

20 Intellectual Property

To be supplied by the RFC Editor.

21 Acknowledgements

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

22 References

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

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

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

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

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

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

[ISO-11578] ISO (International Organization for Standardization), ISO/IEC 11578:1996. "Information technology - Open Systems Interconnection - Remote Procedure Call (RPC)."

[RR] J. Slein, E.J. Whitehead Jr., J. Davis, G. Clemm, C. Fay, J. Crawford, "WebDAV Redirect References." Internet Draft (work in progress) draft-ietf-webdav-redirectref-protocol-02. Xerox, UC Irvine, CourseNet, Rational, FileNet, IBM. December, 1999.

23 Authors' Addresses

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

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

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

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

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

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

24 Appendices

24.1 Appendix 1: Extensions to the WebDAV Document Type Definition

```
<!--============== XML Elements from Section 14 ================-->
<!ELEMENT segment (#PCDATA)>
<!--============== Property Elements from Section 13 ===========-->
<!ELEMENT bindings ((href, segment)*)>
<!ELEMENT resourceid (href)>
```