Network Working Group
Internet-Draft
Updates: 2518 (if approved)
Expires: September 8, 2004

G. Clemm
IBM
J. Crawford
IBM Research
J. Reschke
greenbytes
J. Whitehead
U.C. Santa Cruz
March 10, 2004

Binding Extensions to Web Distributed Authoring and Versioning (WebDAV)
draft-ietf-webdav-bind-04

Status of this Memo

By submitting this Internet-Draft, I certify that any applicable patent or other IPR claims of which I am aware have been disclosed, and any of which I become aware will be disclosed, in accordance with RFC 3667.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on September 8, 2004.

Copyright Notice

Abstract

This specification defines bindings, and the BIND method for creating multiple bindings to the same resource. Creating a new binding to a resource causes at least one new URI to be mapped to that resource. Servers are required to insure the integrity of any bindings that

they allow to be created.

Table of Contents

1. Introduction

This specification extends 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 [RFC2616] and WebDAV [RFC2518] 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.

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.

This specification is organized as follows. Section 1.1 defines terminology used in the rest of the specification, while Section 2 overviews bindings. Section 3 defines the new properties needed to support multiple bindings to the same resource. Section 4 specifies the BIND method, used to create multiple bindings to the same resource. Section 5 specifies the UNBIND method, used to remove a binding to a resource. Section 6 specifies the REBIND method, used to move a binding to another collection.

1.1 Terminology

The terminology used here follows and extends that in the WebDAV Distributed Authoring Protocol specification [RFC2518].

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

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

- o Element names use the "DAV:" namespace.
- o Element ordering is irrelevant.
- o Extension elements/attributes (elements/attributes not already defined as valid child elements) may be added anywhere, except when explicitly stated otherwise.

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 URI makes it possible to submit HTTP protocol requests to the resource using the URI.

Path Segment

Informally, the characters found between slashes ("/") in a URI. Formally, as defined in section 3.3 of [RFC2396].

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 a collection C (accessible through the URI http://www.example.com/CollX), a path segment S (equal to "foo.html"), and a resource R, then creating the binding C: (S -> R) makes it possible to use the URI http://www.example.com/CollX/foo.html to access R.

Collection

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

Internal Member URI

The URI that identifies an internal member of a collection, and that consists of the URI for the collection, followed by a slash character ('/'), followed by the path segment of the binding for that internal member.

1.2 Rationale for Distinguishing Bindings from URI Mappings

In [RFC2518], the state of a collection is defined as containing a list of internal member URIs. If there are multiple mappings to a collection, then the state of the collection is different when you refer to it via a different URI. This is undesirable, since ideally a collection's membership should remain the same, independent of which

URI was used to reference it.

The notion of binding is introduced to separate the final segment of a URI from its parent collection's 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. The authors of this specification anticipate and recommend that future revisions of [RFC2518] will update the definition of the state of a collection to correspond to the definition in this document.

1.3 Method Preconditions and Postconditions

A "precondition" of a method describes the state on the server that must be true for that method to be performed. A "postcondition" of a method describes the state on the server that must be true after that method has completed. If a method precondition or postcondition for a request is not satisfied, the response status of the request MUST be either 403 (Forbidden) if the request should not be repeated because it will always fail, or 409 (Conflict) if it is expected that the user might be able to resolve the conflict and resubmit the request.

In order to allow better client handling of 403 and 409 responses, a distinct XML element type is associated with each method precondition and postcondition of a request. When a particular precondition is not satisfied or a particular postcondition cannot be achieved, the appropriate XML element MUST be returned as the child of a top-level DAV:error element in the response body, unless otherwise negotiated by the request. In a 207 Multi-Status response, the DAV:error element would appear in the appropriate DAV:responsedescription element.

2. Overview of Bindings

Bindings are part of the state of a collection. They define the internal members of the collection, and the names of those internal members.

Bindings are added and removed by a variety of existing HTTP methods. A method that creates a new resource, such as PUT, COPY, and MKCOL, adds a binding. A method that deletes a resource, such as DELETE, removes a binding. A method that moves a resource (e.g. MOVE) both adds a binding (in the destination collection) and removes a binding (in the source collection). The BIND method introduced here provides a mechanism for adding a second binding to an existing resource. There is no difference between an initial binding added by PUT, COPY, or MKCOL, and additional bindings added with BIND.

It would be very undesirable if one binding could be destroyed as a side effect of operating on the resource through a different binding. In particular, the removal of one binding to a resource (e.g. with a DELETE or a MOVE) MUST NOT disrupt another binding to that resource, e.g. by turning that binding into a dangling path segment. The server MUST NOT reclaim system resources after removing one binding, while other bindings to the resource remain. In other words, the server MUST maintain the integrity of a binding.

2.1 Bindings to Collections

Bindings to collections can result in loops, which servers MUST detect when processing "Depth: infinity" requests. 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 7 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 CollY is created for collection C1 in the figure below. It immediately becomes possible to access resource R1 using the URI /CollY/x.gif and to access resource R2 using the URI /CollY/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 /CollY/x.gif and /CollY/y.jpg.

```
+-------------------------+
| Root Collection         |
|  bindings:              |
|  CollX          CollY   |
+-------------------------+
    |            /
    |           /
    |          /
+------------------+
| Collection C1    |
| bindings:        |
| x.gif     y.jpg  |
+------------------+
    |          \
    |           \
    |            \
+-------------+   +-------------+
| Resource R1 |   | Resource R2 |
+-------------+   +-------------+
```

2.2 URI Mappings Created by a new Binding

Suppose a binding from "Binding-Name" to resource R to be added to a collection, C. Then if C-MAP is the set of URIs 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.

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

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

then the following new mappings to R are introduced:

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

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 are introduced.

For example, 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.example.com/A/1/myself
http://www.example.com/A/1/myself/myself
   ...
```

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

```
http://www.example.com/A/1/myself/foo.html
http://www.example.com/A/1/myself/myself/foo.html
   ...
```

2.3 COPY and Bindings

As defined in Section 8.8 of [RFC2518], 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 final segment.

The following figure 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 URI-X. 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 URI-X and the new resource (although other URI mappings may also have been created).

```
URI-1   URI-2    URI-3                          URI-X
   |       |        |                             |
   |       |        |   <---- URI Mappings ---->  |
   |       |        |                             |
+---------------------+                +------------------------+
|     Resource R      |                |     Resource R'        |
+---------------------+                +------------------------+
```

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 [RFC2518] 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.

If a COPY request causes an existing resource to be updated, the bindings to that resource MUST be unaffected by the COPY request. Using the preceding example, suppose that a COPY request is issued to URI-X for resource R', with the Destination header set to URI-2. The content and dead properties of resource R would be updated to be a copy of those of resource R', but the mappings from URI-1, URI-2, and URI-3 to resource R remain unaffected. If because of multiple bindings to a resource, more than one source resource updates a single destination resource, the order of the updates is server defined.

If a COPY request would cause a new resource to be created as a copy of an existing resource, and that COPY request has already created a copy of that existing resource, the COPY request instead creates another binding to the previous copy, instead of creating a new resource.

2.4 DELETE and Bindings

When there are multiple bindings to a resource, a DELETE applied to that resource MUST NOT remove any bindings to that resource other than the one identified by the request URI. For example, suppose the collection identified by the URI "/a" has a binding named "x" to a resource R, and another collection identified by "/b" has a binding named "y" to the same resource R. Then a DELETE applied to "/a/x" removes the binding named "x" from "/a" but MUST NOT remove the binding named "y" from "/b" (i.e. after the DELETE, "/y/b" continues to identify the resource R). In particular, although Section 8.6.1 of [RFC2518] 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", a server that supports the binding protocol MUST NOT follow this requirement.

When DELETE is applied to a collection, it MUST NOT modify the membership of any other collection that is not itself a member of the collection being deleted. For example, if both "/a/.../x" and "/b/.../y" identify the same collection, C, then applying DELETE to "/a" MUST NOT delete an internal member from C or from any other collection that is a member of C, because that would modify the membership of "/b".

If a collection supports the UNBIND method (see Section 5), a DELETE of an internal member of a collection MAY be implemented as an UNBIND request. In this case, applying DELETE to a Request-URI has the effect of removing the binding identified by the final segment of the Request-URI from the collection identified by the Request-URI minus

its final segment. Although [RFC2518] allows a DELETE to be a non-atomic operation, when the DELETE operation is implemented as an UNBIND, the operation 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.

2.5 MOVE and Bindings

When MOVE is applied to a resource, the other bindings to that resource MUST be unaffected, and if the resource being moved is a collection, the bindings to any members of that collection MUST be unaffected. Also, if MOVE is used with Overwrite:T to delete an existing resource, the constraints specified for DELETE apply.

If the destination collection of a MOVE request supports the REBIND method (see Section 6), a MOVE of a resource into that collection MAY be implemented as a REBIND request. Although [RFC2518] allows a MOVE to be a non-atomic operation, when the MOVE operation is implemented as a REBIND, the operation is atomic. In particular, applying a MOVE to a Request-URI and a Destination URI has the effect of removing a binding to a resource (at the Request-URI), and creating a new binding to that resource (at the Destination URI).

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 URI-X. After successful completion of the MOVE operation, a new binding has been created which creates the URI mapping between URI-X and resource R. 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. If resource R were a collection, old URI-3 based mappings to members of R would have been removed, and new URI-X based mappings to members of R would have been created.

```
>> Before Request:

 URI-1   URI-2    URI-3
   |       |        |
   |       |        |      <---- URI Mappings
   |       |        |
+---------------------+
|     Resource R      |
+---------------------+
```

```
>> After Request:

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

Although [RFC2518] allows a MOVE on a collection to be a non-atomic operation, a MOVE implemented as a REBIND 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.

2.6 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 resources might have identical contents and properties, but not be the same resource (e.g. an update to one resource does not affect the other resource).

The REQUIRED DAV:resource-id property defined in Section 3.1 is a resource identifier, which MUST be unique across all resources for all time. If the values of DAV:resource-id 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:resource-id property is created, and its value assigned, when the resource is created. The value of DAV:resource-id 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:resource-id property.

Any method that creates a new resource MUST assign a new, unique value to its DAV:resource-id property. For example, a PUT or a COPY that creates a new resource must assign a new, unique value to the DAV:resource-id property of that new resource.

On the other hand, any method that affects an existing resource MUST NOT change the value of its DAV:resource-id property. For example, a PUT or a COPY that updates an existing resource must not change the value of its DAV:resource-id 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 the DAV:resource-id property.

2.7 Discovering the Bindings to a Resource

An OPTIONAL DAV:parent-set property on a resource provides a list of the bindings that associate a collection and a URI segment with that resource. If the DAV:parent-set property exists on a given resource, it MUST contain a complete list of all bindings to that resource that the client is authorized to see. When deciding whether to support the DAV:parent-set property, server implementers / administrators should balance the benefits it provides against the cost of maintaining the property and the security risks enumerated in Sections 8.4 and 8.5.

3. Properties

The bind feature introduces the following properties for a resource.

A DAV:allprop PROPFIND request SHOULD NOT return any of the properties defined by this document. This allows a binding server to perform efficiently when a naive client, which does not understand the cost of asking a server to compute all possible live properties, issues a DAV:allprop PROPFIND request.

3.1 DAV:resource-id Property

The DAV:resource-id property is a REQUIRED property that enables clients to determine whether two bindings are to the same resource. The value of DAV:resource-id is a URI, and may use any registered URI scheme that guarantees the uniqueness of the value across all resources for all time (e.g. the opaquelocktoken: scheme defined in [RFC2518]).

```
<!ELEMENT resource-id (href)>
```

3.2 DAV:parent-set Property

The DAV:parent-set property is an OPTIONAL property that enables clients to discover what collections contain a binding to this resource (i.e. what collections have that resource as an internal member). It contains an of href/segment pair for each collection that has a binding to the resource. The href identifies the collection, and the segment identifies the binding name of that resource in that collection.

A given collection MUST appear only once in the DAV:parent-set for any given binding, even if there are multiple URI mappings to that collection. For example, if collection C1 is mapped to both /CollX and /CollY, and C1 contains a binding named "x.gif" to a resource R1,

then either [/CollX, x.gif] or [/CollY, x.gif] can appear in the DAV:parent-set of R1, but not both. But if C1 also had a binding named "y.gif" to R1, then there would be two entries for C1 in the DAV:binding-set of R1 (i.e. either both [/CollX, x.gif] and [/CollX, y.gif] or alternatively, both [/CollY, x.gif] and [/CollY, y.gif]).

```
<!ELEMENT parent-set (parent)*>
<!ELEMENT parent (href, segment)>
<!ELEMENT segment (#PCDATA)>
```

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

4. BIND Method

The BIND method modifies the collection identified by the Request-URI, by adding a new binding from the segment specified in the BIND body to the resource identified in the BIND body.

If a server cannot guarantee the integrity of the binding, the BIND request MUST fail. Note that 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 make it inaccessible 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. The precondition DAV:cross-server-binding is defined below for cases where servers fail cross-server BIND requests because they cannot guarantee the integrity of cross-server bindings.

By default, if there already is a binding for the specified segment in the collection, 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 [RFC2518].

Marshalling:

The request MAY include an Overwrite header.

The request body MUST be a DAV:bind XML element.

```
<!ELEMENT bind (segment, href)>
```

If the request succeeds, the server MUST return 201 (Created) when a new binding was created and 200 (OK) when an existing binding was replaced.

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

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

Preconditions:

(DAV:bind-into-collection): The Request-URI MUST identify a collection.

(DAV:bind-source-exists): The DAV:href element MUST identify a resource.

(DAV:binding-allowed): The resource identified by the DAV:href supports multiple bindings to it.

(DAV:cross-server-binding): If the resource identified by the DAV:href element in the request body is on another server from the collection identified by the request-URI, the server MUST support cross-server bindings.

(DAV:name-allowed): The name specified by the DAV:segment is available for use as a new binding name.

(DAV:can-overwrite): If the collection already contains a binding with the specified path segment, and if an Overwrite header is included, the value of the Overwrite header MUST be "T".

(DAV:cycle-allowed): If the DAV:href element identifies a collection, and if the request-URI identifies a collection that is a member of that collection, the server MUST support cycles in the URI namespace.

(DAV:locked-update-allowed): If the collection identified by the Request-URI is write-locked, then the appropriate token MUST be specified in an If request header.

(DAV:locked-overwrite-allowed): If the collection already contains a binding with the specified path segment, and if that binding is protected by a write-lock, then the appropriate token MUST be

specified in an If request header.

Postconditions:

(DAV:new-binding): The collection MUST have a binding that maps the segment specified in the DAV:segment element in the request body, to the resource identified by the DAV:href element in the request body.

4.1 Example: BIND

>> Request:

```
BIND /CollY HTTP/1.1
Host: www.example.com
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx

<?xml version="1.0" encoding="utf-8" ?>
<D:bind xmlns:D="DAV:">
   <D:segment>bar.html</D:segment>
   <D:href>http://www.example.com/CollX/foo.html</D:href>
</D:bind>
```

>> Response:

```
HTTP/1.1 201 Created
```

The server added a new binding to the collection, "http://www.example.com/CollY", associating "bar.html" with the resource identified by the URI "http://www.example.com/CollX/foo.html". Clients can now use the URI "http://www.example.com/CollY/bar.html", to submit requests to that resource.

5. UNBIND Method

The UNBIND method modifies the collection identified by the Request-URI, by removing the binding identified by the segment specified in the UNBIND body.

Once a resource is unreachable by any URI mapping, the server MAY reclaim system resources associated with that resource. If UNBIND 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.

Marshalling:

The request body MUST be a DAV:unbind XML element.

```
<!ELEMENT unbind (segment)>
```

If the request succeeds, the server MUST return 200 (OK) when the binding was successfully deleted.

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

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

Preconditions:

(DAV:unbind-from-collection): The Request-URI MUST identify a collection.

(DAV:unbind-source-exists): The DAV:segment element MUST identify a binding in the collection identified by the Request-URI.

(DAV:locked-update-allowed): If the collection identified by the Request-URI is write-locked, then the appropriate token MUST be specified in the request.

(DAV:protected-url-deletion-allowed): If the binding identified by the segment is protected by a write-lock, then the appropriate token MUST be specified in the request.

Postconditions:

(DAV:binding-deleted): The collection MUST NOT have a binding for the segment specified in the DAV:segment element in the request body.

5.1 Example: UNBIND

```
>> Request:

UNBIND /CollX HTTP/1.1
Host: www.example.com
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx

<?xml version="1.0" encoding="utf-8" ?>
<D:unbind xmlns:D="DAV:">
   <D:segment>foo.html</D:segment>
</D:unbind>

>> Response:

HTTP/1.1 200 OK
```

The server removed the binding named "foo.html" from the collection, "http://www.example.com/CollX". A request to the resource named "http://www.example.com/CollX/foo.html" will return a 404 (Not Found) response.

6. REBIND Method

The REBIND method removes a binding to a resource from one collection, and adds a binding to that resource into another collection. It is effectively an atomic form of a MOVE request.

Marshalling:

The request MAY include an Overwrite header.

The request body MUST be a DAV:rebind XML element.

```
<!ELEMENT rebind (segment, href)>
```

If the request succeeds, the server MUST return 201 (Created) when a new binding was created and 200 (OK) when an existing binding was replaced.

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

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

Preconditions:

(DAV:rebind-into-collection): The Request-URI MUST identify a collection.

(DAV:rebind-source-exists): The DAV:href element MUST identify a resource.

(DAV:binding-allowed): The resource identified by the DAV:href supports multiple bindings to it.

(DAV:cross-server-binding): If the resource identified by the DAV:href element in the request body is on another server from the collection identified by the request-URI, the server MUST support cross-server bindings.

(DAV:name-allowed): The name specified by the DAV:segment is available for use as a new binding name.

(DAV:can-overwrite): If the collection already contains a binding with the specified path segment, and if an Overwrite header is included, the value of the Overwrite header MUST be "T".

(DAV:cycle-allowed): If the DAV:href element identifies a collection, and if the request-URI identifies a collection that is a member of that collection, the server MUST support cycles in the URI namespace.

(DAV:locked-update-allowed): If the collection identified by the Request-URI is write-locked, then the appropriate token MUST be specified in the request.

(DAV:protected-url-modification-allowed): If the collection identified by the Request-URI already contains a binding with the specified path segment, and if that binding is protected by a write-lock, then the appropriate token MUST be specified in the request.

(DAV:locked-source-collection-update-allowed): If the collection identified by the parent collection prefix of the DAV:href URI is write-locked, then the appropriate token MUST be specified in the request.

(DAV:protected-source-url-deletion-allowed): If the DAV:href URI is protected by a write lock, then the appropriate token MUST be specified in the request.

Postconditions:

(DAV:new-binding): The collection MUST have a binding that maps the segment specified in the DAV:segment element in the request body, to the resource that was identified by the DAV:href element in the request body.

(DAV:binding-deleted): The URL specified in the DAV:href element in the request body MUST NOT be mapped to a resource.

6.1 Example: REBIND

```
>> Request:

REBIND /CollX HTTP/1.1
Host: www.example.com
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx

<?xml version="1.0" encoding="utf-8" ?>
<D:rebind xmlns:D="DAV:">
   <D:segment>foo.html</D:segment>
   <D:href>http://www.example.com/CollY/bar.html</D:href>
</D:rebind>

>> Response:

HTTP/1.1 200 OK
```

The server added a new binding to the collection, "http://www.example.com/CollX", associating "foo.html" with the resource identified by the URI "http://www.example.com/CollY/bar.html", and removes the binding named "bar.html" from the collection identified by the URI "http://www.example.com/CollY". Clients can now use the URI "http://www.example.com/CollX/foo.html" to submit requests to that resource, and requests on the URI "http://www.example.com/CollY/bar.html" will fail with a 404 (Not Found) response.

7. Additional Status Codes

7.1 208 Already Reported

The 208 (Already Reported) status code can be used inside a DAV:propstat response element to avoid enumerating the internal members of multiple bindings to the same collection repeatedly. For each binding to a collection inside the request's scope, only one will be reported with a 200 status, while subsequent DAV:response

elements for all other bindings will use the 208 status, and no DAV:response elements for their descendants are included.

Note that the 208 status will only occur for "Depth: infinity" requests, and that it is of particular importance when the multiple collection bindings cause a bind loop as discussed in Section 2.2.

A client can request the DAV:resourceid property in a PROPFIND request to guarantee that they can accurately reconstruct the binding structure of a collection with multiple bindings to a single resource.

For backward compatibility with clients not aware of the 208 status code appearing in multistatus response bodies, it SHOULD NOT be used unless the client has signalled support for this specification using the "DAV" request header (see Section 8.2). Instead, a 506 status should be returned when a binding loop is discovered. This allows the server to return the 506 as the top level return status, if it discovers it before it started the response, or in the middle of a multistatus, if it discovers it in the middle of streaming out a multistatus response.

7.1.1 Example: PROPFIND by bind-aware client

For example, consider a PROPFIND request on /Coll (bound to collection C), where the members of /Coll are /Coll/Foo (bound to resource R) and /Coll/Bar (bound to collection C).

>> Request:

```
PROPFIND /Coll/ HTTP/1.1
Host: www.example.com
Depth: infinity
DAV: bind
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.example.com/Coll/</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.example.com/Coll/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.example.com/Coll/Bar</D:href>
      <D:propstat>
         <D:prop>
            <D:displayname>Loop Demo</D:displayname>
         </D:prop>
         <D:status>HTTP/1.1 208 Already Reported</D:status>
      </D:propstat>
   </D:response>
</D:multistatus>
```

7.1.2 Example: PROPFIND by non-bind-aware client

In this example, the client isn't aware of the 208 status code introduced by this specification. As the "Depth: infinity" PROPFIND request would cause a loop condition, the whole request is rejected with a 506 status.

```
>> Request:

PROPFIND /Coll/ HTTP/1.1
Host: www.example.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/1.1 506 Loop Detected
```

7.2 506 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". This status indicates that the entire operation failed.

8. Capability discovery

8.1 OPTIONS method

If the server supports bindings, it MUST return the compliance class name "bind" as a field in the "DAV" response header (see [RFC2518], section 9.1) from an OPTIONS request on any resource implemented by that server. A value of "bind" in the "DAV" header MUST indicate that the server supports all MUST level requirements and REQUIRED features specified in this document.

8.2 'DAV' request header

8.2.1 Generic syntax

This specification introduces the 'DAV' request header that allows clients to signal compliance to specific WebDAV features. It has the same syntax as the response header defined in [RFC2518], section 9.1, but MAY be used with any method.

Note that clients MUST NOT submit a specific compliance class name in the request header unless the specification defining this compliance class specifically defines it's semantics for clients.

Note that if a server chooses to vary the result of a request based on values in the "DAV" header, the response either MUST NOT be cacheable or the server MUST mark the response accordingly using the "Vary" header (see [RFC2616], section 14.44).

8.2.2 Client compliance class 'bind'

Clients SHOULD signal support for all MUST level requirements and REQUIRED features by submitting a "DAV" request header containing the compliance class name "bind". In particular, the client MUST understand the 208 status code defined in Section 7.1.

9. Security Considerations

This section is provided to make WebDAV implementors 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.

9.1 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.

9.2 Bind Loops

Although bind 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".

9.3 Bindings, and Denial of Service

Denial of service attacks were already possible by posting URIs 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.

9.4 Private Locations May Be Revealed

If the DAV:parent-set 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:parent-set property on the resource. Moving a binding may reveal its new location to anyone with access to DAV:parent-set on its resource.

9.5 DAV:parent-set and Denial of Service

If the server maintains the DAV:parent-set 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.

10. Internationalization Considerations

All internationalization considerations mentioned in [RFC2518] also apply to this document.

11. IANA Considerations

All IANA considerations mentioned in [RFC2518] also apply to this document.

12. Acknowledgements

This draft is the collaborative product of the authors and Tyson Chihaya, Jim Davis, Chuck Fay and Judith Slein. This draft has benefited from thoughtful discussion by Jim Amsden, Peter Carlson, Steve Carter, Ken Coar, Ellis Cohen, Dan Connolly, Bruce Cragun, Spencer Dawkins, Mark Day, Rajiv Dulepet, David Durand, Stefan Eissing, 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 other members of the WebDAV working group.

Normative References

[RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.

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

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

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

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

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E. and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Third Edition)", W3C REC-xml-20040204, February 2004, <http://www.w3.org/TR/2004/REC-xml-20040204>.

Authors' Addresses

Geoffrey Clemm
IBM
20 Maguire Road
Lexington, MA 02421

EMail: geoffrey.clemm@us.ibm.com

Jason Crawford
IBM Research
P.O. Box 704
Yorktown Heights, NY 10598

EMail: ccjason@us.ibm.com

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

EMail: julian.reschke@greenbytes.de

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

EMail: ejw@cse.ucsc.edu

Appendix A. Change Log (to be removed by RFC Editor before publication)

A.1 Since draft-ietf-webdav-bind-02

Add and resolve issues "2.3_COPY_SHARED_BINDINGS" and "2.3_MULTIPLE_COPY". Add issue "5.1_LOOP_STATUS" and proposed resolution, but keep it open. Add issues "ED_references" and "4_507_status". Started work on index. Rename document to "Binding Extensions to Web Distributed Authoring and Versioning (WebDAV)". Rename "References" to "Normative References". Close issue "ED_references". CLose issue "4_507_status".

A.2 Since draft-ietf-webdav-bind-03

Add and close issues "9.2_redirect_loops", "ED_authors" and "ED_updates". Add section about capability discovery (DAV header). Close issues "5.1_LOOP_STATUS". Add and resolve new issue "5.1_506_STATUS_STREAMING". Update XML spec reference. Add issue "locking" and resolve as invalid.

Appendix B. Resolved issues (to be removed by RFC Editor before publication)

Issues that were either rejected or resolved in this version of this document.

B.1 ED_updates

Type: edit

<http://lists.w3.org/Archives/Public/w3c-dist-auth/2003OctDec/0306.html>

julian.reschke@greenbytes.de (2003-12-30): Shouldn't the BIND spec be labelled as "updating" RFC2518 and/or RFC3253?

julian.reschke@greenbytes.de (2004-01-05): It seems that there was consensus to say that BIND does update RFC2518, while there's no consensus about the relationship to RFC3253. As this is a minor editorial question, I propose to just say "updated 2518" and to close the issue.

Resolution (2004-01-09): State "updates 2518".

B.2 ED_authors

Type: edit

julian.reschke@greenbytes.de (2004-01-02): Ensure contact information for all original authors is still correct, and that the authors in fact support the current draft.

Resolution (2004-01-05): J. Slein removed as author and added to Acknowledgments.

B.3 locking

Type: change

<http://www.xmpp.org/ietf-logs/webdav@ietf.xmpp.org/2004-03-01.html>

lisa@xythos.com (2004-03-02): (concerns about the behaviour of multiple bindings to the same resource when the resource was locked via WebDAV LOCK on one of it's bindings).

Resolution (2004-03-06): No issue here: given two URIs "a" and "b" mapped to the same resource, and a WebDAV LOCK that was requested for "a", attempts to modify the lockable state of "b" will fail unless the lock-token is presented. This is because the resource itself is locked, not only it's URI "a". See RFC2518, section 6.

B.4 5.1_LOOP_STATUS

Type: change

julian.reschke@greenbytes.de (2002-10-11): Should the 506 status in a PROPFIND be handled differently?

geoffrey.clemm@us.ibm.com (2003-08-03): Use new 208 status to report cycles in PROPFIND.

julian.reschke@greenbytes.de (2003-11-16): Proposal: a) import DAV request header definition from rfc2518bis (note that the definition in the latest draft probably needs some more work) b) define a DAV compliance class for the BIND spec c) clarify that 208 should only be returned when the client specifies compliance to the BIND spec in the PROPFIND request (otherwise fail the complete request).

Resolution (2004-01-02): Define DAV compliance class "bind", define DAV request header, clarify that 208 must only be used when the

client signals that it knows about this specification.

B.5 5.1_506_STATUS_STREAMING

Type: change

julian.reschke@greenbytes.de (2004-01-12): Take a server that allows Depth: infinity upon PROPFIND. In general, the only way to implement this efficiently is to *stream* the multistatus response. However this means that the server needs to decide about the HTTP status code to return *prior* to actually doing the recursion. Therefore, requiring the server to return a 506 status in case there's a bind loop somewhere below the request URI will not work. An obvious alternative would be to allow the 506 to be returned inside the DAV:status element for the resource the server refuses to recurse into. Of course that would mean that the client essentially would get a partly incorrect picture of the server state. However, I don't think there's any way to avoid that (except for rejecting all Depth: infinity PROPFINDs, like many servers already do).

Resolution (2004-03-06): Allow 506 inside multistatus.

B.6 9.2_redirect_loops

Type: edit

julian.reschke@greenbytes.de (2004-01-09): Avoid confusion with REDIRECT. Propose rename to "Bind Loops".

Resolution (2004-01-09): Do the rename.

Index

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

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