Internet Engineering Task Force (IETF)
Request for Comments: 5842
Category: Experimental
ISSN: 2070-1721

G. Clemm
IBM
J. Crawford
IBM Research
J. Reschke, Ed.
greenbytes
J. Whitehead
U.C. Santa Cruz
April 2010

Binding Extensions to Web Distributed Authoring and Versioning (WebDAV)

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 ensure the integrity of any bindings that they allow to be created.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5842.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction

This specification extends the WebDAV Distributed Authoring Protocol ([RFC4918]) 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, for example, 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 [RFC4918] 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 a new URI for submitting requests to an existing resource available. 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 [RFC4918].

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 notational convention, using the rules defined in Section 17 of [RFC4918].

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 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 [RFC3986].

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.

Binding Integrity

The property of a binding that says that:

* the binding continues to exist, and

* the identity of the resource identified by that binding does not change,

unless an explicit request is executed that is defined to delete that binding (examples of requests that delete a binding are DELETE, MOVE, and -- defined later on -- UNBIND and REBIND).

1.2. Method Preconditions and Postconditions

See Section 16 of [RFC4918] for the definitions of "precondition" and "postcondition".

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. It is permissible, however, for future method definitions (e.g., a DESTROY method) to have semantics that explicitly remove all bindings and/or immediately reclaim system resources.

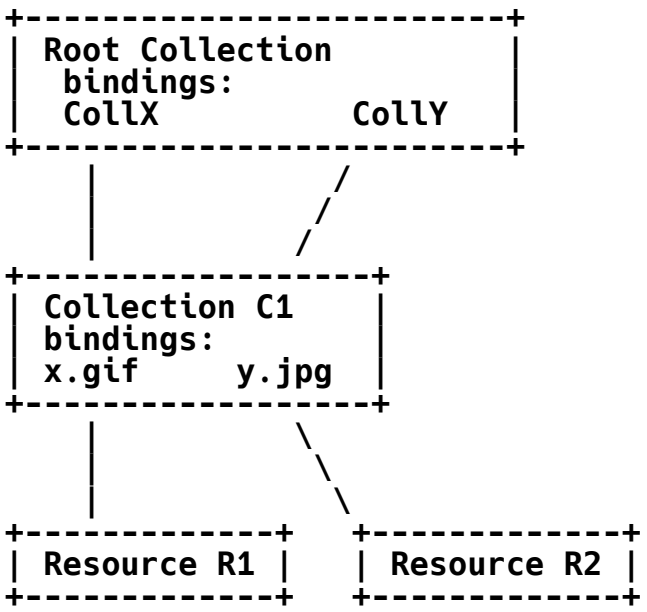
> Note: the collection model described herein is not compatible with systems in which resources inherit properties based solely on the access path, as the ability to create additional bindings will cause a single resource to appear as member of several different collections at the same time.

2.1. Bindings to Collections

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 they 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.



2.1.1. Bind Loops

Bindings to collections can result in loops ("cycles"), 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. For instance, a PROPFIND can still succeed if the server uses the new status code 208 (Already Reported) defined in Section 7.1.

However, the 508 (Loop Detected) status code is defined in Section 7.2 for use in contexts where an operation is terminated because a loop was encountered.

Support for loops is OPTIONAL: servers MAY reject requests that would lead to the creation of a bind loop (see DAV:cycle-allowed precondition defined in Section 4).

2.2. URI Mappings Created by a New Binding

Suppose a binding from "Binding-Name" to resource R is 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
...

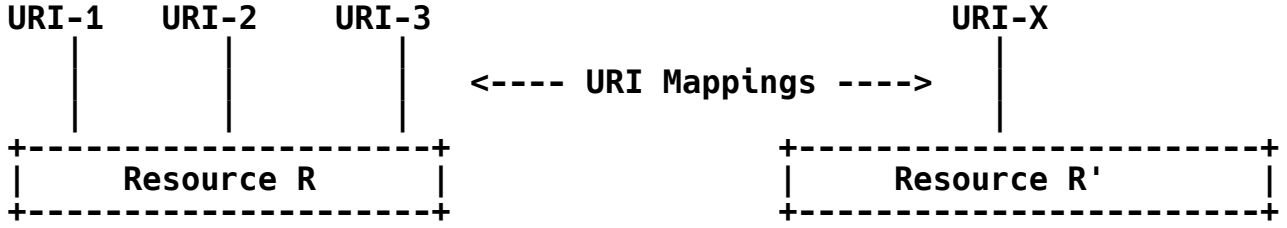
and the following infinite number of additional mappings to R are introduced:

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 9.8 of [RFC4918], 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 that creates at least the URI mapping between URI-X and the new resource (although other URI mappings may also have been created).



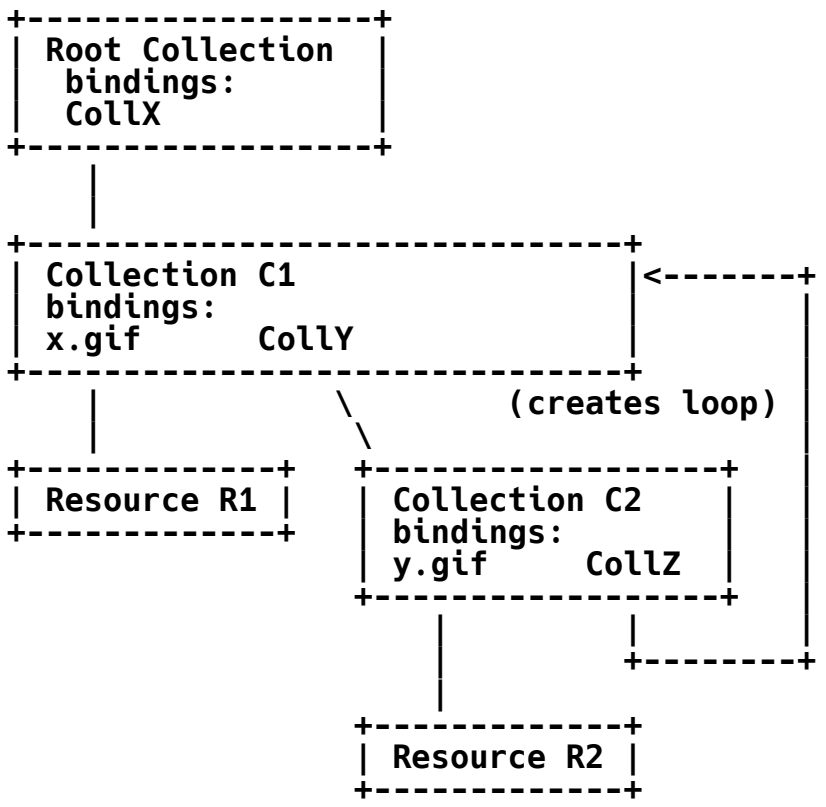
It might be thought that a COPY request with "Depth: 0" on a collection would duplicate its bindings, since bindings are part of the collection's state. This is not the case, however. The definition of Depth in [RFC4918] makes it clear that a "Depth: 0" request does not apply to a collection's members. 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 (see Section 2.3.2 for an example).

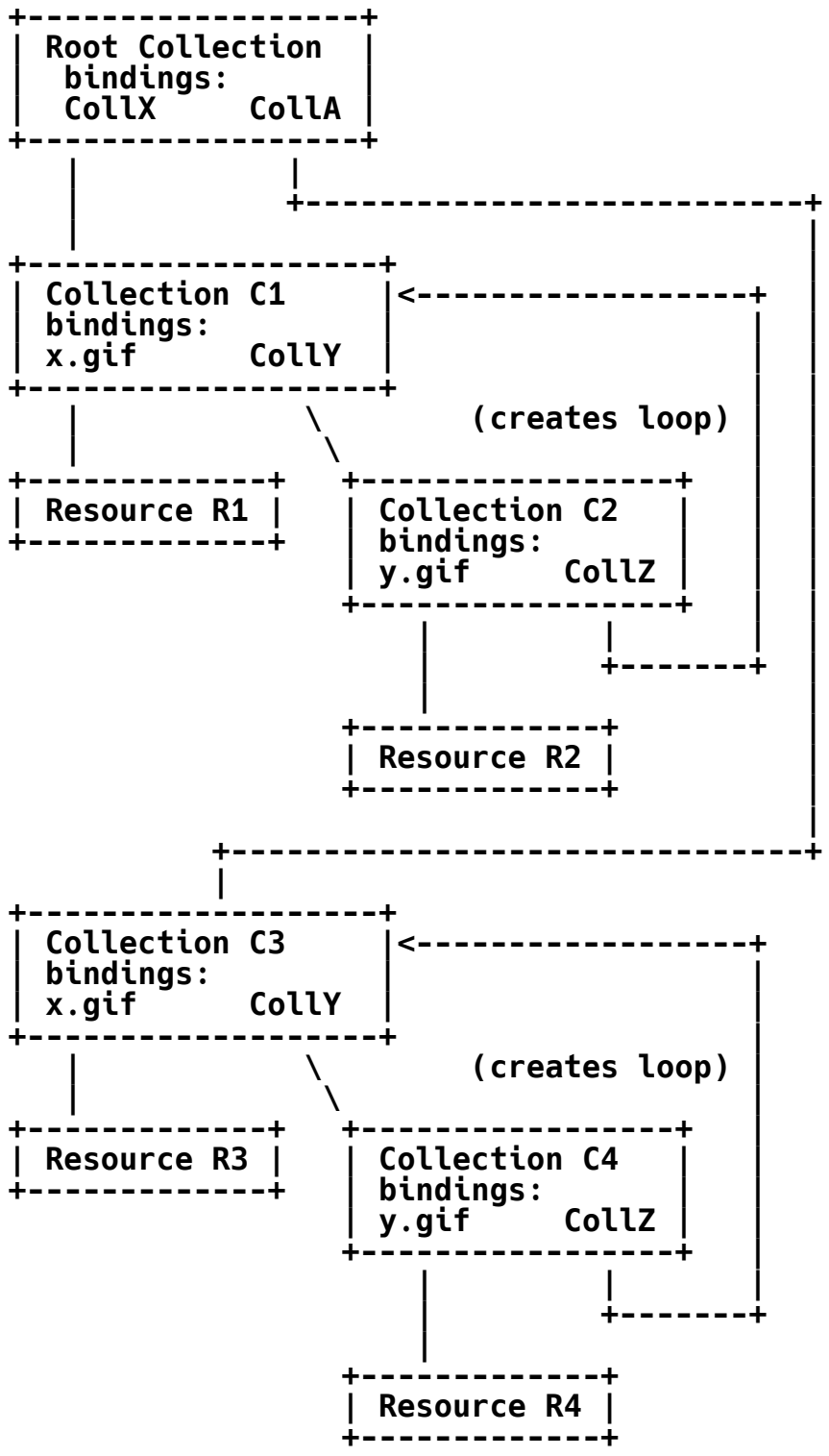
If a COPY request would cause a new resource to be created as a copy of an existing resource, and that COPY request has already created a copy of that existing resource, the COPY request instead creates another binding to the previous copy, instead of creating a new resource (see Section 2.3.3 for an example).

2.3.1. Example: COPY with "Depth: infinity" in Presence of Bind Loops

As an example of how COPY with "Depth: infinity" would work in the presence of bindings, consider the following collection:



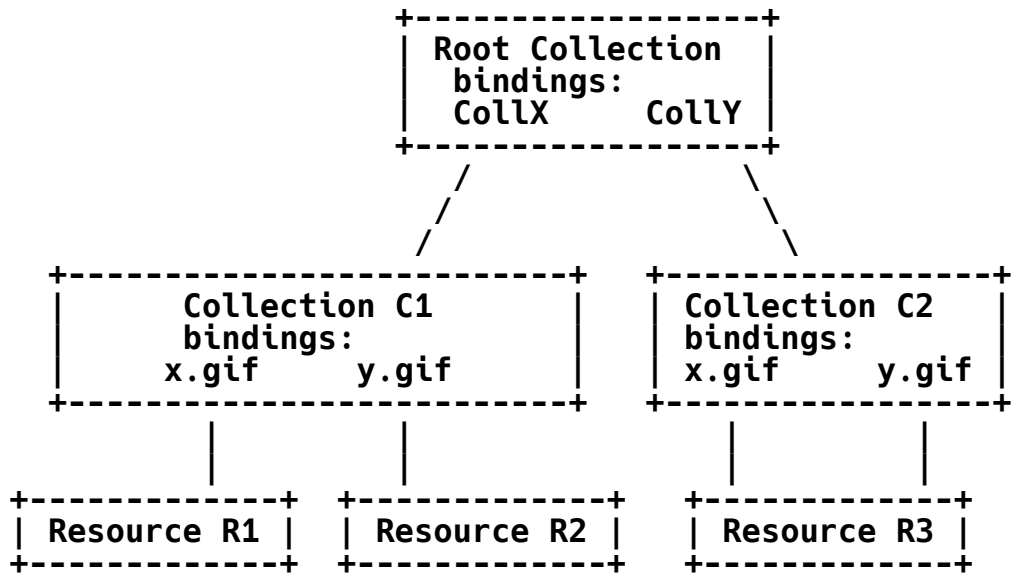
If a COPY request with "Depth: infinity" is submitted to /CollX, with a destination of /CollA, the outcome of the copy operation is that a copy of the tree is replicated to the target /CollA:


Root Collection
bindings:
CollX CollA
Collection C1
bindings:
x.gif CollY
(creates loop)
Resource R1
Collection C2
bindings:
y.gif CollZ
Resource R2
Collection C3
bindings:
x.gif CollY
(creates loop)
Resource R3
Collection C4
bindings:
y.gif CollZ
Resource R4

Note that the same would apply for more complex loops.

2.3.2. Example: COPY Updating Multiple Bindings

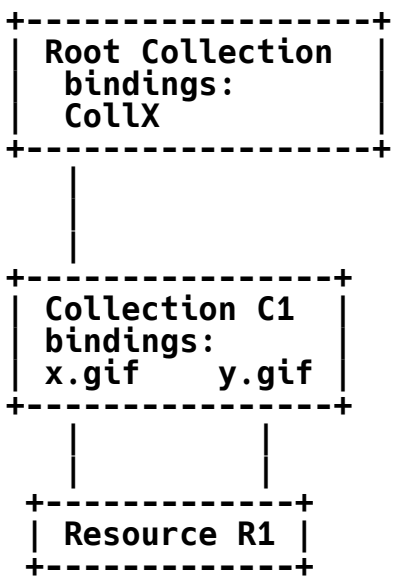
Given the following collection hierarchy:



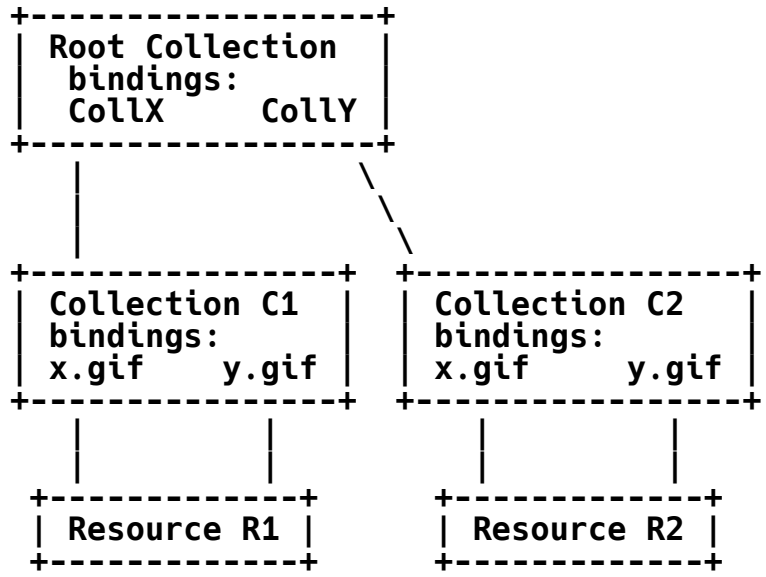
A COPY of /CollX with "Depth: infinity" to /CollY will not result in a changed hierarchy, and Resource R3 will be updated with the content of either Resource R1 or Resource R2.

2.3.3. Example: COPY with "Depth: infinity" with Multiple Bindings to a Leaf Resource

Given the following collection hierarchy:



A COPY of /CollX with "Depth: infinity" to /CollY results in the following collection hierarchy:



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).

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 [RFC4918] 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.

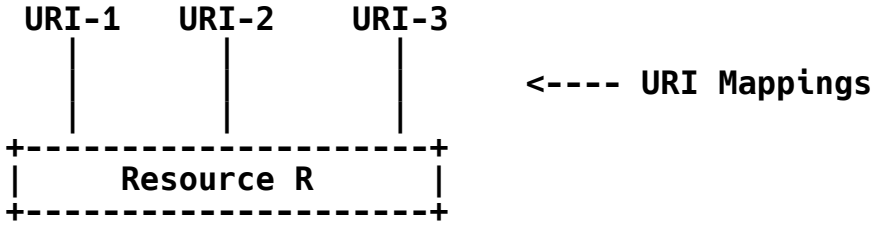
If the destination collection of a MOVE request supports the REBIND method (see Section 6), a MOVE of a resource into that collection MAY be implemented as a REBIND request. Although [RFC4918] allows a MOVE to be a non-atomic operation, when the MOVE operation is implemented as a REBIND, the operation 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). Even when the Request-URI identifies a collection, the MOVE operation involves only removing one binding to that collection and adding another.

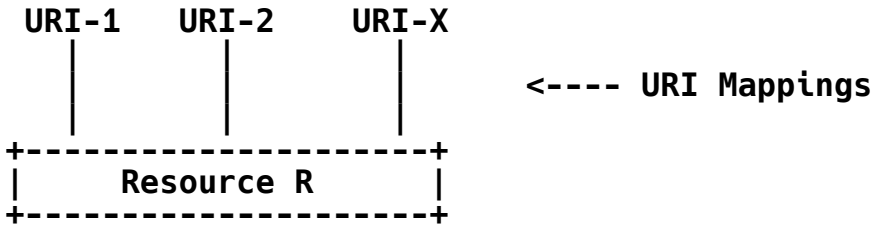
2.5.1. Example: Simple MOVE

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 that 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:



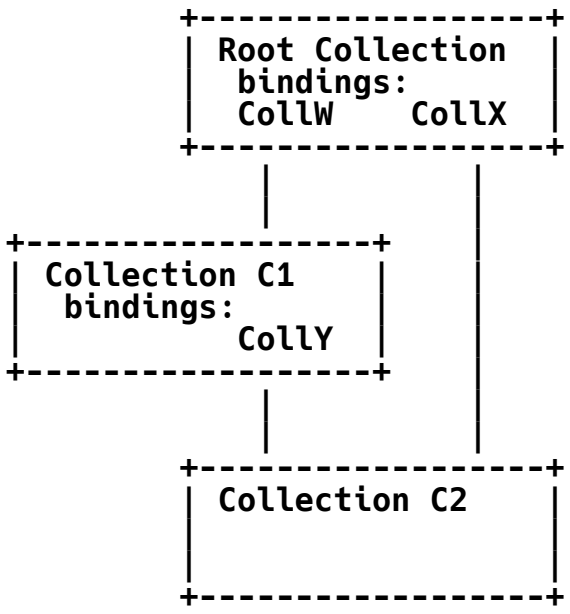
>> After Request:



2.5.2. Example: MOVE Request Causing a Bind Loop

Note that in the presence of collection bindings, a MOVE request can cause the creation of a bind loop.

Consider the top-level collections C1 and C2 with URIs "/CollW/" and "/CollX/". C1 also contains an additional binding named "CollY" to C2:

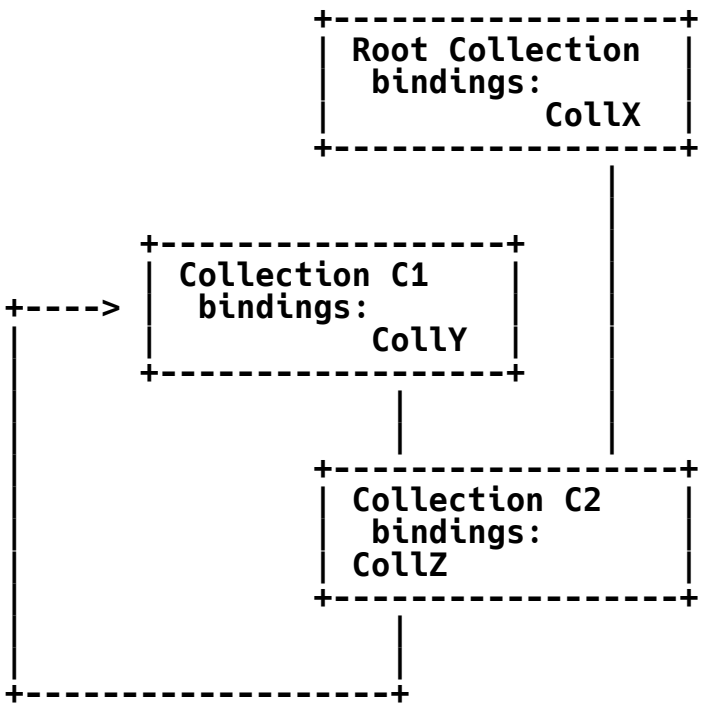


In this case, the MOVE request below would cause a bind loop:

>> Request:

```
MOVE /CollW HTTP/1.1
Host: example.com
Destination: /CollX/CollZ
```

If the request succeeded, the resulting state would be:



2.6. PROPFIND and Bindings

Consistent with [RFC4918], the value of a dead property MUST be independent of the number of bindings to its host resource or of the path submitted to PROPFIND. On the other hand, the behavior for each live property depends on its individual definition (for example, see [RFC3744], Section 5, Paragraph 2 for a case where the value is independent of its path and bindings, and [RFC4918], Section 8.8 for a discussion about the live properties DAV:getetag and DAV: getlastmodified, which may behave differently).

2.7. 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 character by character, 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 applied to a null resource, COPY (when not overwriting an existing target) and CHECKIN (see [RFC3253], Section 4.4) must assign a new, unique value to the DAV:resource-id property of the new resource they create.

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

2.8. 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 12.4 and 12.5.

3. Properties

The bind feature introduces the properties defined below.

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 urn:uuid: URN namespace defined in [RFC4122] or the opaquelocktoken: URI scheme defined in [RFC4918]).

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

3.2. DAV:parent-set Property

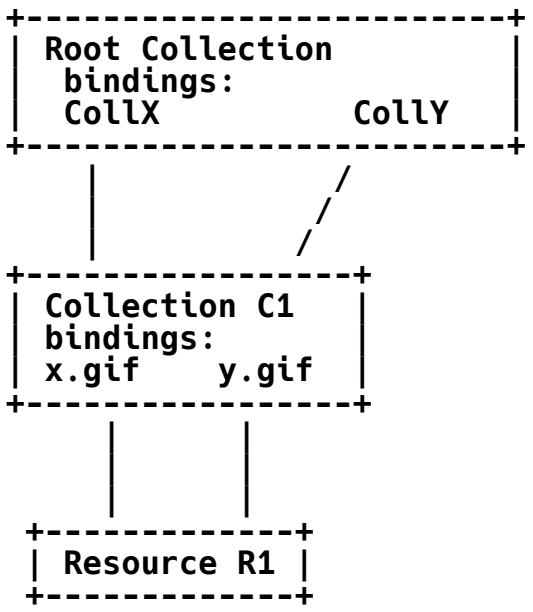
The DAV:parent-set property is an OPTIONAL property that enables clients to discover what collections contain a binding to this resource (i.e., what collections have that resource as an internal member). It contains an href/segment pair for each collection that has a binding to the resource. The href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

```
<!ELEMENT parent-set (parent)*>
<!ELEMENT parent (href, segment)>
<!ELEMENT segment (#PCDATA)>
<!-- PCDATA value: segment, as defined in Section 3.3 of
     [RFC3986] -->
```

3.2.1. Example for DAV:parent-set Property

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:parent-set of R1 (i.e., both [/CollX, x.gif] and [/CollX, y.gif] or, alternatively, both [/CollY, x.gif] and [/CollY, y.gif]).



In this case, one possible value for the DAV:parent-set property on "/CollX/x.gif" would be:

```
<parent-set xmlns="DAV:">
  <parent>
    <href>/CollX</href>
    <segment>x.gif</segment>
  </parent>
  <parent>
    <href>/CollX</href>
    <segment>y.gif</segment>
  </parent>
</parent-set>
```

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 the 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 10.6 of [RFC4918].

If a BIND request fails, the server state preceding the request MUST be restored. This method is unsafe and idempotent (see [RFC2616], Section 9.1).

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) or 204 (No Content) 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 (servers that do not support cross-server bindings can use this condition code to signal the client exactly why the request failed).

(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 (servers that do not support cycles can use this condition code to signal the client exactly why the request failed).

(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: application/xml; charset="utf-8"
Content-Length: 172

<?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
Location: http://www.example.com/CollY/bar.html
```

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.

If an UNBIND request fails, the server state preceding the request MUST be restored. This method is unsafe and idempotent (see [RFC2616], Section 9.1).

Marshalling:

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

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

If the request succeeds, the server MUST return 200 (OK) or 204 (No Content) 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.

(DAV:lock-deleted): If the internal member URI of the binding specified by the Request-URI and the DAV:segment element in the request body was protected by a write lock at the time of the request, that write lock must have been deleted by the request.

5.1. Example: UNBIND

```
>> Request:

UNBIND /CollX HTTP/1.1
Host: www.example.com
Content-Type: application/xml; charset="utf-8"
Content-Length: 117

<?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 a collection, and adds a binding to that resource into the collection identified by the Request-URI. The request body specifies the binding to be added (segment) and the old binding to be removed (href). It is effectively an atomic form of a MOVE request, and MUST be treated the same way as MOVE for the purpose of determining access permissions.

If a REBIND request fails, the server state preceding the request MUST be restored. This method is unsafe and idempotent (see [RFC2616], Section 9.1).

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) or 204 (No Content) 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: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 (servers that do not support cross-server bindings can use this condition code to signal the client exactly why the request failed).

(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 (servers that do not support cycles can use this condition code to signal the client exactly why the request failed).

(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.

(DAV:lock-deleted): If the URL specified in the DAV:href element in the request body was protected by a write lock at the time of the request, that write lock must have been deleted by the request.

6.1. Example: REBIND

>> Request:

```
REBIND /CollX HTTP/1.1
Host: www.example.com
Content-Type: application/xml; charset="utf-8"
Content-Length: 176

<?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:

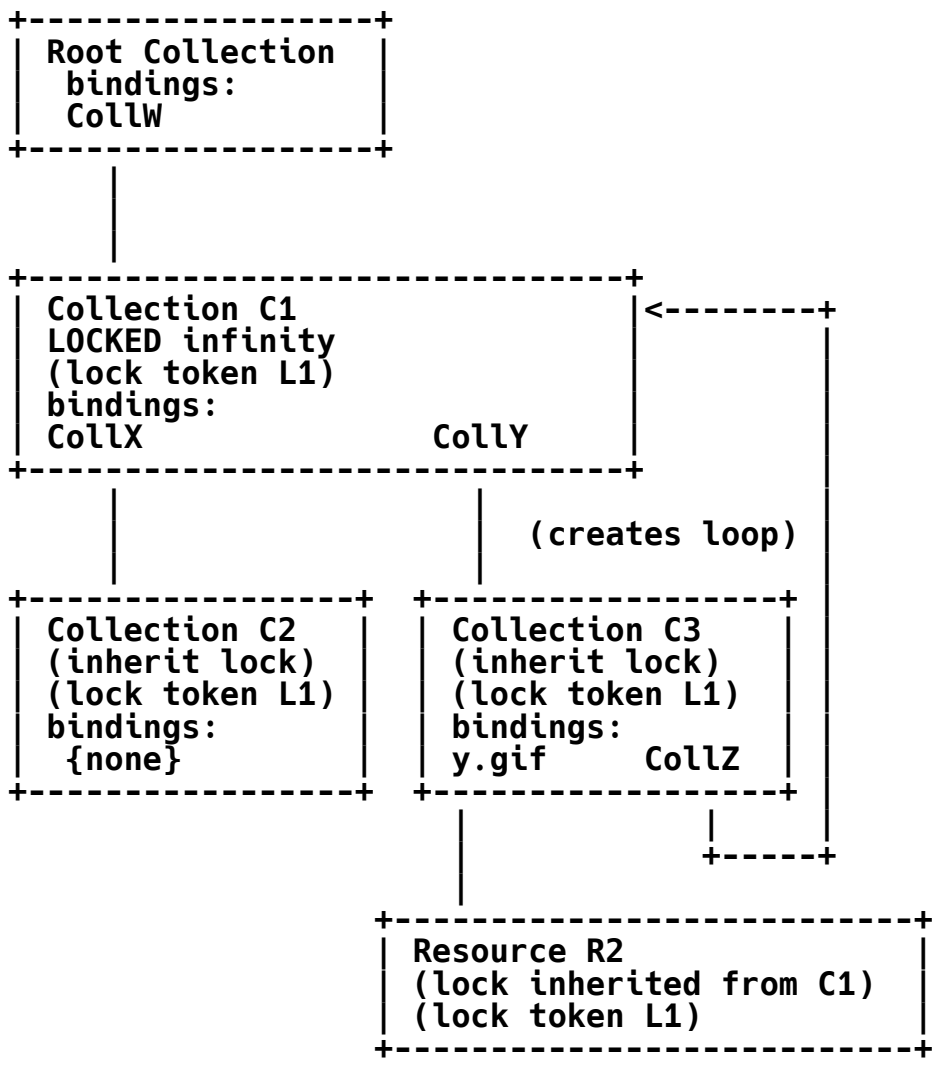
```
HTTP/1.1 200 OK
```

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.

6.2. Example: REBIND in Presence of Locks and Bind Loops

To illustrate the effects of locks and bind loops on a REBIND operation, consider the following collection:



(where L1 is "urn:uuid:f92d4fae-7012-11ab-a765-00c0ca1f6bf9").

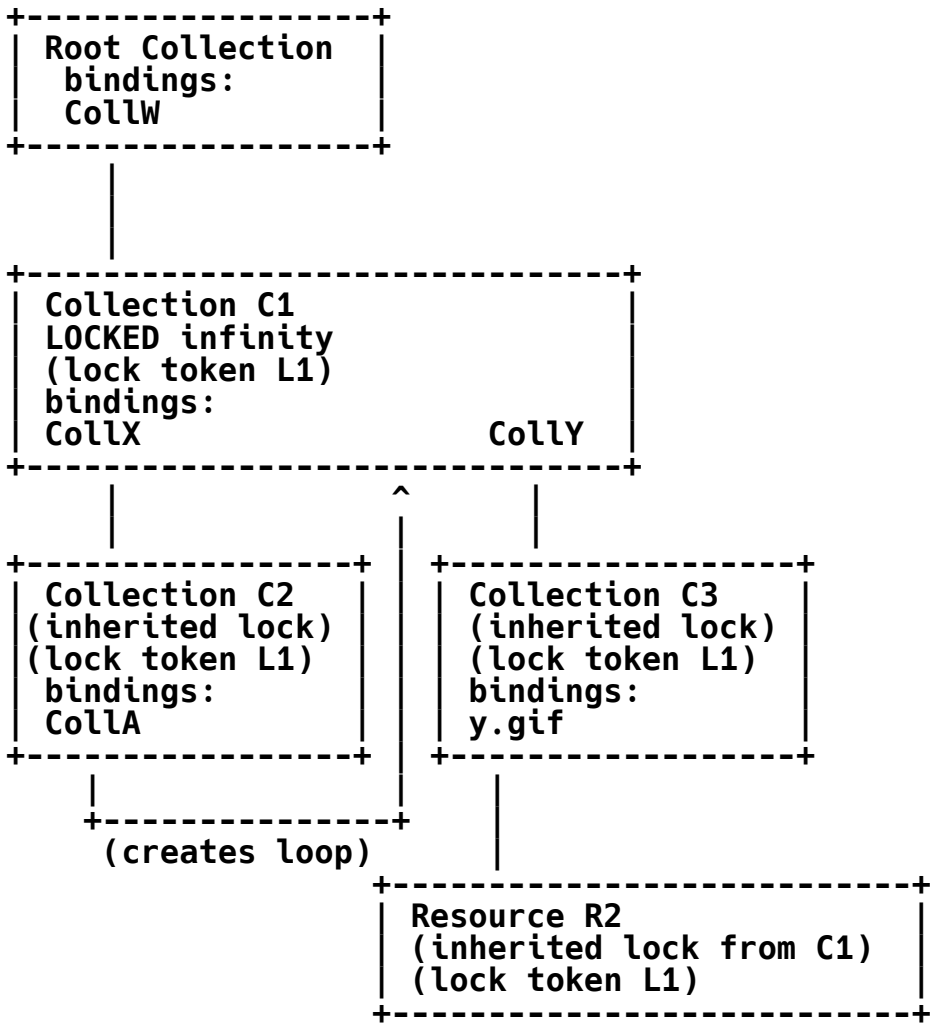
Note that the binding between CollZ and C1 creates a loop in the containment hierarchy. Servers are not required to support such loops, though the server in this example does.

The REBIND request below will remove the segment "CollZ" from C3 and add a new binding from "CollA" to the collection C2.

```
REBIND /CollW/CollX HTTP/1.1
Host: www.example.com
If: (<urn:uuid:f92d4fae-7012-11ab-a765-00c0ca1f6bf9>)
Content-Type: application/xml; charset="utf-8"
Content-Length: 152

<?xml version="1.0" encoding="utf-8" ?>
<D:rebind xmlns:D="DAV:">
  <D:segment>CollA</D:segment>
  <D:href>/CollW/CollY/CollZ</D:href>
</D:rebind>
```

The outcome of the REBIND operation is:



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:resource-id 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 signaled support for this specification using the "DAV" request header (see Section 8.2). Instead, a 508 status should be returned when a binding loop is discovered. This allows the server to return the 508 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: application/xml; charset="utf-8"
Content-Length: 152

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

```
   >> Response:

   HTTP/1.1 207 Multi-Status
   Content-Type: application/xml; charset="utf-8"
   Content-Length: 1241

   <?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:resource-id>
             <D:href
   >urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf8</D:href>
           </D:resource-id>
         </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:resource-id>
             <D:href
   >urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf9</D:href>
           </D:resource-id>
         </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:resource-id>
             <D:href
   >urn:uuid:f81d4fae-7dec-11d0-a765-00a0c91e6bf8</D:href>
           </D:resource-id>
         </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 508 status.

```
>> Request:

PROPFIND /Coll/ HTTP/1.1
Host: www.example.com
Depth: infinity
Content-Type: application/xml; charset="utf-8"
Content-Length: 125

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

>> Response:

HTTP/1.1 508 Loop Detected
```

7.2. 508 Loop Detected

The 508 (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 [RFC4918], Section 10.1) 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

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. Relationship to Locking in WebDAV

Locking is an optional feature of WebDAV ([RFC4918]). The base WebDAV specification and this protocol extension have been designed in parallel, making sure that all features of WebDAV can be implemented on a server that implements this protocol as well.

Unfortunately, WebDAV uses the term "lock-root" inconsistently. It is introduced in Section 6.1 of [RFC4918], point 2, as:

> 2. A resource becomes directly locked when a LOCK request to a URL of that resource creates a new lock. The "lock-root" of the new lock is that URL. If at the time of the request, the URL is not mapped to a resource, a new empty resource is created and directly locked.

On the other hand, [RFC4918], Section 9.10.1 states:

> A LOCK request to an existing resource will create a lock on the resource identified by the Request-URI, provided the resource is not already locked with a conflicting lock. The resource identified in the Request-URI becomes the root of the lock.

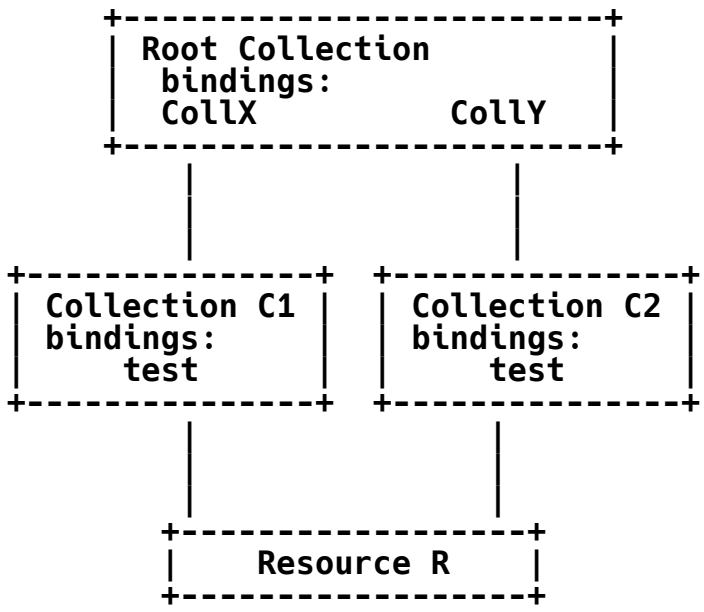
Servers that implement both WebDAV locking and support for multiple bindings MUST use the first interpretation: the lock-root is the URI through which the lock was created, not a resource. This URI, and potential aliases of this URI ([RFC4918], Section 5), are said to be "protected" by the lock.

As defined in the introduction to Section 7 of [RFC4918], write operations that modify the state of a locked resource require that the lock token is submitted with the request. Consistent with WebDAV, the state of the resource consists of the content ("any variant"), dead properties, lockable live properties (item 1), plus, for a collection, all its bindings (item 2). Note that this, by definition, does not depend on the Request-URI to which the write operation is applied (the locked state is a property of the resource, not its URI).

However, the lock-root is the URI through which the lock was requested. Thus, the protection defined in item 3 of the list does not apply to additional URIs that may be mapped to the same resource due to the existence of multiple bindings.

9.1. Example: Locking and Multiple Bindings

Consider a root collection "/", containing the two collections C1 and C2, named "/CollX" and "/CollY", and a child resource R, bound to C1 as "/CollX/test" and bound to C2 as "/CollY/test":



Given a host name of "www.example.com", applying a depth-zero write lock to "/CollX/test" will lock the resource R, and the lock-root of this lock will be "http://www.example.com/CollX/test".

Thus, the following operations will require that the associated lock token is submitted with the "If" request header ([RFC4918], Section 10.4):

- a PUT or PROPPATCH request modifying the content or lockable properties of resource R (as R is locked) -- no matter which URI is used as request target, and

- a MOVE, REBIND, UNBIND, or DELETE request causing "/CollX/test" not to be mapped to resource R anymore (be it addressed to "/CollX" or "/CollX/test").

The following operations will not require submission of the lock token:

- a DELETE request addressed to "/CollY" or "/CollY/test", as it does not affect the resource R, nor the lock-root,

- for the same reason, an UNBIND request removing the binding "test" from collection C2, or the binding "CollY" from the root collection, and

- similarly, a MOVE or REBIND request causing "/CollY/test" not being mapped to resource R anymore.

Note that despite the lock-root being "http://www.example.com/CollX/test", an UNLOCK request can be addressed through any URI mapped to resource R, as UNLOCK operates on the resource identified by the Request-URI, not that URI (see [RFC4918], Section 9.11).

10. Relationship to WebDAV Access Control Protocol

Note that the WebDAV Access Control Protocol has been designed for compatibility with systems that allow multiple URIs to map to the same resource (see [RFC3744], Section 5):

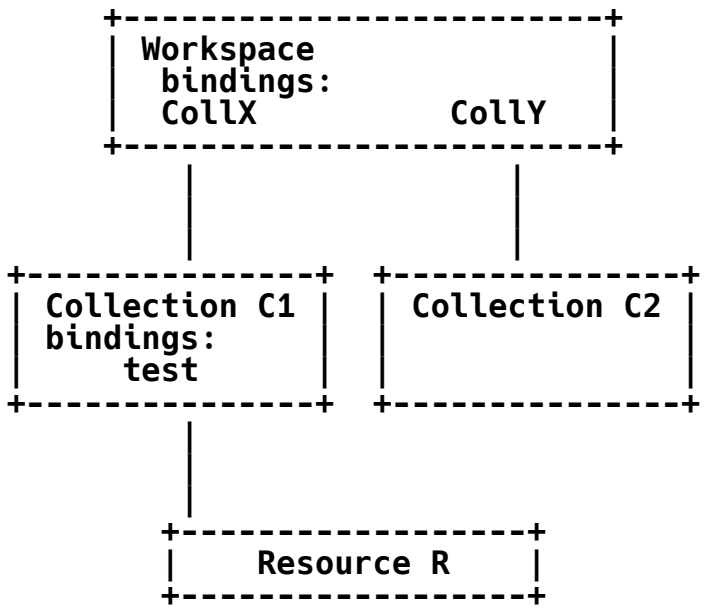
> Access control properties (especially DAV:acl and DAV:inherited-acl-set) are defined on the resource identified by the Request-URI of a PROPFIND request. A direct consequence is that if the resource is accessible via multiple URI, the value of access control properties is the same across these URI.

Furthermore, note that BIND and REBIND behave the same as MOVE with respect to the DAV:acl property (see [RFC3744], Section 7.3).

11. Relationship to Versioning Extensions to WebDAV

Servers that implement Workspaces ([RFC3253], Section 6) and Version-Controlled Collections ([RFC3253], Section 14) already need to implement BIND-like behavior in order to handle UPDATE and UNCHECKOUT semantics.

Consider a workspace "/ws1/", containing the version-controlled, checked-out collections C1 and C2, named "/ws1/CollX" and "/ws1/CollY", and a version-controlled resource R, bound to C1 as "/ws1/CollX/test":



Moving "/ws1/CollX/test" into "/ws1/CollY", checking in C2, but undoing the checkout on C1 will undo part of the MOVE request, thus restoring the binding from C1 to R, but keeping the new binding from C2 to R:

```
>> Request:

MOVE /ws1/CollX/test HTTP/1.1
Host: www.example.com
Destination: /ws1/CollY/test

>> Response:

HTTP/1.1 204 No Content

>> Request:

CHECKIN /ws1/CollY/ HTTP/1.1
Host: www.example.com

>> Response:

HTTP/1.1 201 Created
Cache-Control: no-cache
Location: http://repo.example.com/his/17/ver/42
```

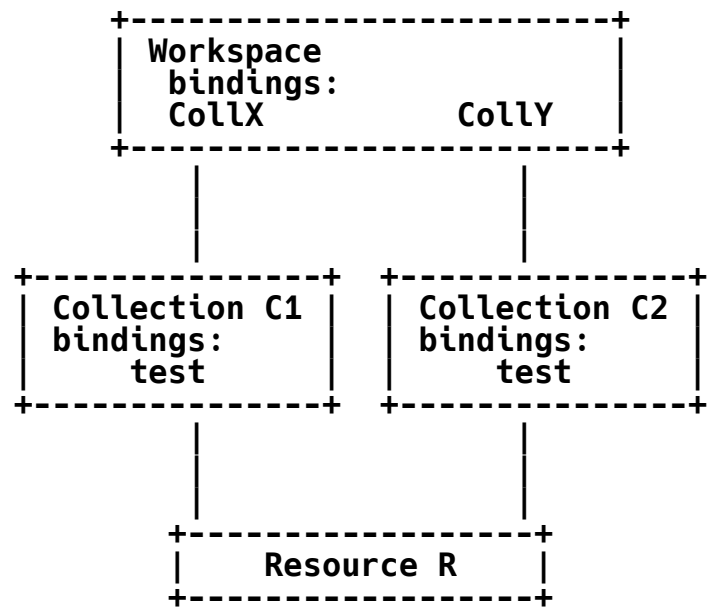
```
>> Request:

UNCHECKOUT /ws1/CollX/ HTTP/1.1
Host: www.example.com

>> Response:

HTTP/1.1 200 OK
Cache-Control: no-cache
```

As a result, both C1 and C2 would have a binding to R:



The MOVE semantics defined in Section 3.15 of [RFC3253] already require that "/ws1/CollX/test" and "/ws1/CollY/test" will have the same version history (as exposed in the DAV:version-history property). Furthermore, the UNCHECKOUT semantics (which in this case is similar to UPDATE, see Section 14.11 of [RFC3253]) require:

> If a new version-controlled member is in a workspace that already has a version-controlled resource for that version history, then the new version-controlled member MUST be just a binding (i.e., another name for) that existing version-controlled resource.

Thus, "/ws1/CollX/test" and "/ws1/CollY/test" will be bindings to the same resource R, and have identical DAV:resource-id properties.

12. Security Considerations

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

All of the security considerations of HTTP/1.1 ([RFC2616], Section 15) and the WebDAV Distributed Authoring Protocol specification ([RFC4918], Section 20) 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.

12.1. 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.

12.2. 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".

12.3. 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.

12.4. 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.

12.5. 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.

13. Internationalization Considerations

All internationalization considerations mentioned in Section 19 of [RFC4918] also apply to this document.

14. IANA Considerations

Section 7 defines the HTTP status codes 208 (Already Reported) and 508 (Loop Detected), which have been added to the HTTP Status Code Registry.

15. Acknowledgements

This document is the collaborative product of the authors and Tyson Chihaya, Jim Davis, Chuck Fay and Judith Slein. It has benefited from thoughtful discussion by Jim Amsden, Peter Carlson, Steve Carter, Ken Coar, Ellis Cohen, Dan Connolly, Bruce Cragun, Cyrus Daboo, Spencer Dawkins, Mark Day, Werner Donne, Rajiv Dulepet, David Durand, Lisa Dusseault, Stefan Eissing, Roy Fielding, Yaron Goland, Joe Hildebrand, Fred Hitt, Alex Hopmann, James Hunt, Marcus Jager, Chris Kaler, Manoj Kasichainula, Rohit Khare, Brian Korver, Daniel LaLiberte, Steve Martin, Larry Masinter, Jeff McAffer, Alexey Melnikov, Surendra Koduru Reddy, Max Rible, Sam Ruby, Bradley Sergeant, Nick Shelness, John Stracke, John Tigue, John Turner, Kevin Wiggen, and other members of the concluded WebDAV working group.

16. References

16.1. Normative References

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

[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.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4918] Dusseault, L., Ed., "HTTP Extensions for Web Distributed Authoring and Versioning (WebDAV)", RFC 4918, June 2007.

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)", W3C REC-xml-20081126, November 2008, <http://www.w3.org/TR/2008/REC-xml-20081126/>.

16.2. Informative References

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

[RFC3744] Clemm, G., Reschke, J., Sedlar, E., and J. Whitehead, "Web Distributed Authoring and Versioning (WebDAV) Access Control Protocol", RFC 3744, May 2004.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.

Index

Authors' Addresses

Geoffrey Clemm
IBM
550 King Street
Littleton, MA 01460

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 (editor)
greenbytes GmbH
Hafenweg 16
Muenster, NW 48155
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