INTERNET-DRAFT
draft-ietf-webdav-bind-00

G. Clemm
Rational Software
J. Crawford
IBM Research
J. Reschke
Greenbytes
J. Slein
Xerox
E.J. Whitehead
U.C. Santa Cruz

Expires April 2, 2002

October 2, 2001

Binding Extensions to 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 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 and WebDAV methods are able to work because there are mappings between URIs and resources. A method is addressed to a URI, and the server follows the mapping from that URI to a resource, applying the method to that resource. Multiple URIs may be mapped to the same resource, but until now there has been no way for clients to create additional URIs mapped to existing resources.

BIND lets clients associate a new URI with an existing WebDAV resource, and this URI can then be used to submit requests to the resource. Since URIs of WebDAV resources are hierarchical, and

correspond to a hierarchy of collections in resource space, the BIND method also has the effect of adding the resource to a collection. As new URIs are associated with the resource, it appears in additional collections.

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 specifies the BIND method, used to create multiple bindings to the same resource. Sections Error! Reference source not found. defines the new properties needed to support multiple bindings to the same resource.

1.1 Terminology

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

URI Mapping

A relation between an absolute URI and a resource. For an absolute URI U and the resource it identifies R, the URI mapping can be thought of as (U => R). Since a resource can represent items that are not network retrievable, as well as those that are, it is possible for a resource to have zero, one, or many URI mappings. Mapping a resource to an "http" scheme URL makes it possible to submit HTTP protocol requests to the resource using the URL.

Path Segment

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

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

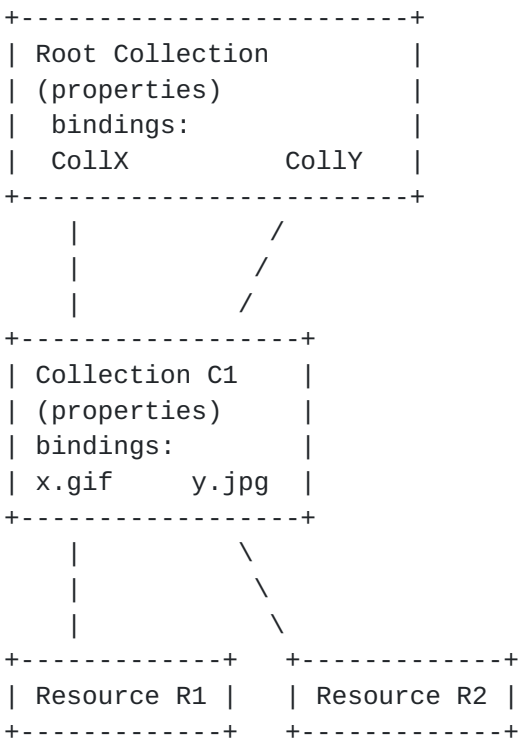
segment. The server MUST NOT reclaim system resources after removing one binding, while other bindings to the resource remain. In other words, the server MUST maintain the integrity of a binding.

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 5 for use in contexts where an operation is terminated because a loop was encountered. Servers MUST allow loops to be created.

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.



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 URI's that were mapped to C before the BIND request, then for each URI "C-URI" in

C-MAP, the URI "C-URI/Binding-Name" is mapped to resource R following the BIND request.

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

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

then the following new mappings to R are introduced:

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

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

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

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

2.3 DELETE and Bindings

The DELETE method was originally defined in [RFC2616]. This section redefines the behavior of DELETE in terms of bindings, an abstraction not available when writing [RFC2616]. [RFC2616] states that "the DELETE method requests that the origin server delete the resource identified by the Request-URI." Because [RFC2616] did not distinguish between bindings and resources, the intent of its definition of DELETE is unclear. The definition presented here is a clarification of the definition in [RFC2616].

The DELETE method requests that the server remove the binding between the resource identified by the Request-URI and the binding name, the last path segment of the Request-URI. The binding MUST be removed from its parent collection, identified by the Request-URI

minus its trailing slash (if present) and final segment.

Once a resource is unreachable by any URI mapping, the server MAY reclaim system resources associated with that resource. If DELETE removes a binding to a resource, but there remain URI mappings to

that resource, the server MUST NOT reclaim system resources associated with the resource.

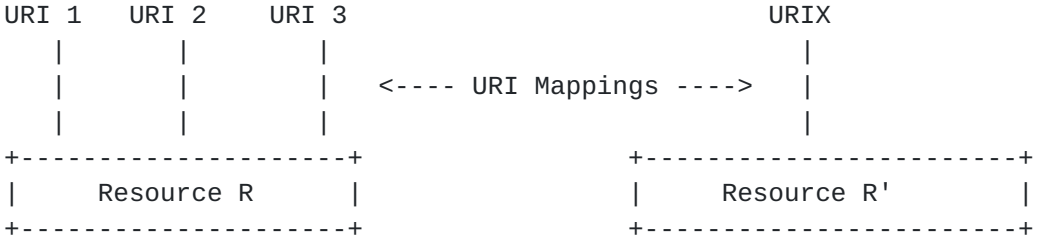
Although [RFC2518] allows a DELETE to be a non-atomic operation, the DELETE operation defined here is atomic. In particular, a DELETE on a hierarchy of resources is simply the removal of a binding to the collection identified by the Request-URI, and so is a single (and therefore atomic) operation.

Section 8.6.1 of [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." Servers that support bindings MUST NOT follow this requirement.

2.4 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 trailing slash (if present) and 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 URIX. After successful completion of the COPY operation, resource R is duplicated to create resource R', and a new binding has been created which creates at least the URI mapping between URIX and the new resource (although other URI mappings may also have been created).



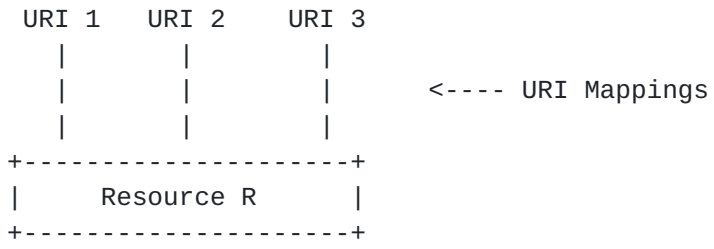
It might be thought that a COPY request with "Depth: 0" on a collection would duplicate its bindings, since bindings are part of the collection's state. This is not the case, however. The definition of Depth in [RFC2518] makes it clear that a "Depth: 0" request does not apply to a collection's members. Consequently, a COPY with "Depth: 0" does not duplicate the bindings contained by the collection.

2.5 MOVE and Bindings

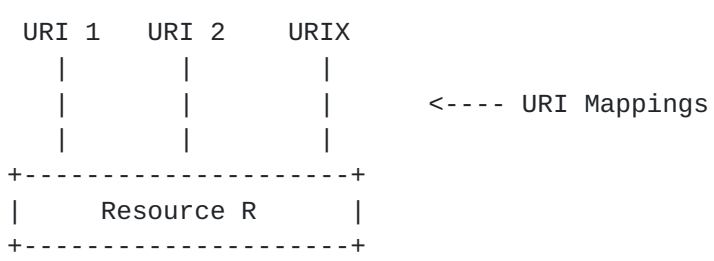
The MOVE method has the effect of creating a new binding to a resource (at the Destination), and removing an existing binding (at the Request-URI). The name of the new binding is the last path segment of the Destination header, and the new binding is added to its parent collection, identified by the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

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

>> Before Request:



>> After Request:



Although [RFC2518] allows a MOVE on a collection to be a non-atomic operation, the MOVE operation defined here MUST be atomic. Even when the Request-URI identifies a collection, the MOVE operation involves only removing one binding to that collection and adding another. There are no operations on bindings to any of its children, so the case of MOVE on a collection is the same as the case of MOVE on a non-collection resource. Both are atomic.

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 that creates a new resource must assign a new, unique value to its DAV:resource-id property. A COPY, since it creates a new resource at the Destination URI, must assign a new, unique value to its DAV:resource-id property.

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 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 its 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 6.4 and 6.5.

3 PROPERTIES

The bind feature introduces the following properties for a resource.

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, y.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. Status code 507 (Cross-server Binding Forbidden) is defined in Section 5.1 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 ANY>
<!ELEMENT bind (segment, href)>
```

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-URL MUST identify a collection.

(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-URL, the server MUST support cross-server bindings.

(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".

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 /coll HTTP/1.1
```

```
Host: www.somehost.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.somehost.com/coll</D:href>
</D:bind>
```

>> Response:

```
HTTP/1.1 201 Created
```

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

5 ADDITIONAL STATUS CODES

5.1 506 Loop Detected

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

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

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

For example, consider a PROPFIND request on /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.somehost.com
Depth: infinity
Content-Type: text/xml; charset="utf-8"
Content-Length: xxx

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

```
     >> Response:

     HTTP/1.1 207 Multi-Status
     Content-Type: text/xml; charset="utf-8"
```

```
Content-Length: xxx

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

6 SECURITY CONSIDERATIONS

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

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

6.1 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.

6.2 Redirect Loops

Although redirect loops were already possible in HTTP 1.1, the introduction of the BIND method creates a new avenue for clients to create loops accidentally or maliciously. If the binding and its target are on the same server, the server may be able to detect

BIND requests that would create loops. Servers are required to detect loops that are caused by bindings to collections during the processing of any requests with "Depth: infinity".

6.3 Bindings, and Denial of Service

Denial of service attacks were already possible by posting URLs that were intended for limited use at heavily used Web sites. The introduction of BIND creates a new avenue for similar denial of service attacks. If cross-server bindings are supported, clients can now create bindings at heavily used sites to target locations that were not designed for heavy usage.

6.4 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.

6.5 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.

7 INTERNATIONALIZATION CONSIDERATIONS

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

8 IANA CONSIDERATIONS

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

9 INTELLECTUAL PROPERTY

The following notice is copied from RFC 2026, Section 10.4, and describes the position of the IETF concerning intellectual property claims made against this document.

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use other technology described in

this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the procedures of the IETF with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF Secretariat.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to practice this standard. Please address the information to the IETF Executive Director.

10 ACKNOWLEDGEMENTS

This draft is the collaborative product of the authors and Tyson Chihaya, Jim Davis, and Chuck Fay. 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, 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.

11 REFERENCES

[RFC2026] S.Bradner, "The Internet Standards Process", RFC 2026, October 1996.

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

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

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

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

February 1999.

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

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

12 AUTHORS' ADDRESSES

Geoffrey Clemm
Rational Software Corporation
20 Maguire Road
Lexington, MA 02173-3104
Email: geoffrey.clemm@rational.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

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

Jim Whitehead
UC Santa Cruz, Dept. of Computer Science
1156 High Street, Santa Cruz, CA 95064
Email: ejw@cse.ucsc.edu