INTERNET-DRAFT
Expires: April 1998

Yaron Y. Goland
Saveen Reddy
Microsoft Corporation
November 6, 1997

WebDAV Tree Operations
draft-ietf-webdav-depth-01.txt

1. Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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

To view the entire list of current Internet-Drafts, please check the "1id-abstracts.txt" listing contained in the Internet-Drafts Shadow Directories on ftp.is.co.za (Africa), ftp.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (US East Coast), or ftp.isi.edu (US West Coast).

2. Abstract

The WebDAV protocol specification [Goland et al., 1997] defines the DELETE, COPY and MOVE methods. However these methods have a scope of a single source resource. It is common for principals to wish to perform a DELETE, COPY or MOVE on a collection and all its internal members. This specification defines the DELETE-TREE, COPY-TREE and MOVE-TREE methods that perform the equivalent of DELETE, COPY and MOVE across a collection and all its progeny.

3. Contents

4. Problem Definition

HTTP is designed such that a single message causes a single action on a single resource. This has proven to be a simple, interoperable, robust mechanism for delivering methods. In addition, in a world where the majority of requests are GETS, it is also a 'fair' arbitrator of server resources. Specifically, as load increases each client suffers degradation in service proportional to the number of requests made.

However clients often wish to perform actions against all internal members of a collection. Currently a client has no choice but to execute each method individually on each member of the collection, in other words, there is no way to instruct a server to recurse through a namespace on behalf of a client.

In many cases forcing the client to perform their own recursive calls is a desirable situation as it maintains the fairness of load distribution. The average HTTP editing server, which handles mostly GETs and PUTs with the occasional COPY or MOVE, is probably better off using non-recursive operations.

However some servers routinely deal with operations on collection, so routinely in fact that they have developed a number of optimizations to allow them to quickly execute an operation against a hierarchy.

A typical example is a copy on write system which can copy an entire hierarchy by putting a single pointer into the server's internal namespace and then tracking when one of the original resources is

changed, thus performing the copy only when required. These servers are unable to take advantage of their optimizations because DAV does not provide a way for a client to tell the server that it intends to execute the copy against an entire hierarchy.

In addition, in some circumstances, it is too expensive for clients to handle recursion themselves. For example, a hand held unit with limited memory, power, and bandwidth, would not be able to deal very well with a simple operation such as deleting a collection. The hand held unit would be required to execute a large number of methods and potentially record a large number of index entries as it recurses through the hierarchy. In such cases fairness takes second place to access.

As such a means is needed for a client to efficiently indicate to a server its desire to execute a single method against a hierarchy.

5. Proposed Solution

The proposed solution is the introduction of three new methods: DELETE-TREE, COPY-TREE, and MOVE-TREE.

The three new methods are not the same as their root methods, DELETE, COPY, and MOVE. For example, a MOVE on a collection has different semantics then MOVE on a single resource.

Clients MUST NOT rely upon the three new methods executing on members of their hierarchies in any particular order and the three new methods are not atomic.

Upon executing the three new methods will perform as much of their assigned task as possible and then return a response specifying what they were able to accomplish and what they failed to do.

So, for example, an attempt to COPY a hierarchy may result in some of the members being copied and some not.

6. Levels of Recursion

As currently defined, all three new methods apply to the full length of the hierarchy. It has been suggested that the number of levels to be recursed should be an option. However no compelling case has been presented for why allowing the depth of recursion to be controlled is a desirable feature. As such this specification errs on the side of simplicity and declares that all three new methods apply to the full hierarchy.

7. Message Headers and Recursion

Any headers on the three new methods MUST be applied to all resources in the scope of the method. For example, an if-match header will have its value applied against every resource in the method's scope and will cause the method to fail if the header fails

to match properly.

[Ed. Note: No, this isn't an error. Think about it, if you put an 'if-match: *' what are you after? I think that putting propagation rules are just going to complicate things beyond reason. Look at the typical case 'I only want to copy this collection if its membership has changed or if the value of its members have changed.' The best an e-tag could give you is detection of membership change, not if the member's values have changed. I say leave well enough alone and just propagate everything.]

8. Lock Tokens and *-TREE Methods

If a resource, source or destination, within scope of the *-TREE method is locked in such a way as to prevent the successful execution of the *-TREE method, then the lock token for that resource MUST be submitted with the *-TREE request in the State-Token request header.

9. DELETE-TREE Method

9.1. Request

The DELETE-TREE method is only meaningful on a collection. If used on a non-collection the DELETE-TREE MUST be treated as a DELETE.

DELETE-TREE instructs that the collection specified in the request-URI, the records of its external member resources, and all its internal member resources, are to be deleted.

If any member can not be deleted then all of the member's progeny MUST NOT be deleted, so as to maintain the namespace.

Any headers included with DELETE-TREE MUST be applied in processing every resource to be deleted. In this case, a header of special interest is the DESTROY header which specifies the method to be used to delete all resources in the scope of the DELETE-TREE.

When the DELETE-TREE method has completed processing it MUST return a consistent namespace. Please refer to [Goland et al., 1997] for a full definition of a consistent namespace.

9.2. Response

The response SHOULD be a multi-status response that describes the result of the DELETE-TREE on each effected resource.

[Editor's Note: The response to a TREE method could potentially be huge, larger than a client may want or need to deal with. It has been suggested that clients be given the ability to tell the server they only want to get back a response code, not a response body.

Thoughts?]

9.3. Response Codes

415 Conflict - This can be used to indicate that some unspecified problem has occurred which makes it impossible to delete a particular resource. The most common scenario is that a new internal member was added to a collection while a DELETE-TREE was running and thus the collection can not be deleted.

9.4. Example

```
DELETE-TREE  /container/ HTTP/1.1
Host: www.foo.bar
Destroy: <http://www.ietf.org/standards/dav/NoUndelete>


HTTP/1.1 207 Multi-Response
Content-Type: text/xml
Content-Length: xxxxx

<?namespace href = "http://www.ietf.org/standards/dav/" As = "d"?>
<d:multiresponse>
  <d:response>
       <d:href>http://www.foo.bar/container/resource1</d:href>
       <d:href>http://www.foo.bar/container/resource2</d:href>
       <d:status>HTTP/1.1 200 Success</d:status>
  </d:response>
  <d:response>
       <d:href>http://www.foo.bar/container/</d:href>
       <d:status>HTTP/1.1 418 Method Failure</d:status>
  </d:response>
  <d:response>
       <d:href>http://www.foo.bar/container/resource3</d:href>
       <d:status>HTTP/1.1 412 Precondition Failed</d:status>
  </d:response>
</d:multiresponse>
```

In this example the attempt to delete http://www.foo.bar/container/resource3 failed. Given that there is only one precondition, one can figure out that the failure was caused the inability of the system to meet the requirement of the Destroy header. Normally however, the client will not know exactly what precondition caused the failure.

The result is that container wasn't deleted because of the failure to delete container/resource3.

[Ed-Note: To state the obvious, do we want to provide information on which precondition actually failed? This is not the panacea it might

seem as the failure may have occurred for multiple reasons and listing a bunch of headers may or may not be useful. Besides, the reality is, nobody every pays attention to error codes. There are

really only two error codes in the world "It worked" or "Something Went Wrong."]

10. COPY-TREE Method

10.1. Request

The COPY-TREE method is only meaningful on a collection. If used on a non-collection the COPY-TREE MUST be treated as a COPY.

COPY-TREE instructs that the collection specified in the Request-URI, the records of its external member resources, and all its internal member resources, are to be copied to a location relative to the Destination header.

Any headers included with COPY-TREE are to be applied in processing every resource to be copied.

The exception to this rule is the Destination header. This header only specifies the destination for the Request-URI. When applied to members of the collection specified in the request-URI the value of Destination is to be modified to reflect the current location in the hierarchy. So, if the request-URI is "a" and the destination is "b" then when a/c/d is processed it MUST use a destination of b/c/d.

When the COPY-TREE method has completed processing it MUST have created a consistent namespace at the destination. Thus if it is not possible to COPY a collection with internal members, the internal members may still be copied but a collection will have to be created at the destination to contain them.

Please refer to the definition of COPY in section XYZ of [Goland et Al., 1997] for the rules on merging members and properties of source collections with pre-existing collections at the destination.

10.2. Response

The response is a multi-status response that describes the result of the COPY-TREE on each effected resource. The response is given for the resource that was to be copied, not the resource that was created as a result of the copy. In other words, each entry indicates if the copy on the resource specified in the href succeeded or failed and why.

The exception to this rule is for errors that occurred on the destination. For example, if the destination was locked the response would indicate the destination URL and a 416 "Locked" error.

10.3. Example

```
COPY-TREE  /container/ HTTP/1.1
Host: www.foo.bar
Destination: http://www.foo.bar/othercontainer/
Enforce-Live-Properties: *

HTTP/1.1 207 Multiresponse
Content-Type: text/xml
Content-Length: xxxxx

<?namespace href = "http://www.ietf.org/standards/dav/" As = "d"?>
<d:multiresponse>
  <d:response>
       <d:href>http://www.foo.bar/container/resource1</d:href>
       <d:href>http://www.foo.bar/container/resource2</d:href>
       <d:href>http://www.foo.bar/container/</d:href>
       <d:href>http://www.foo.bar/container/R2/D2</d:href>
       <d:status>HTTP/1.1 201 Created</d:status>
  </d:response>
  <d:response>
       <d:href>http://www.foo.bar/container/R2/</d:href>
       <d:status>HTTP/1.1 415 Precondition Failed</d:status>
  </d:response>
</d:multiresponse>
```

In this example most of the resources, along with the container, were copied successfully. However the container R2 failed, most likely due to a problem with enforcing live properties. R2's member D3 was successfully copied. As a result a collection was created at www.foo.bar/othercontainer/R2 to contain D2.

11. MOVE-TREE Method

11.1. Request

The MOVE-TREE method is only meaningful on a collection. If used on a non-collection the MOVE-TREE MUST be treated as a MOVE.

MOVE-TREE instructs that the collection specified in the Request-URI, the records of its external member resources, and all its internal member resources, are to be moved to a location relative to the Destination header.

Any headers included with MOVE-TREE are to be applied in processing every resource to be moved.

The exception to this rule is the Destination header. The behavior of this header is the same as given for COPY-TREE.

When the MOVE-TREE method has completed processing it MUST have created a consistent namespace on both the source and destination, creating collections at the source or destination as necessary.

As specified in the definition of MOVE, a MOVE of a collection over another collection causes the destination collection and all its members to be deleted.

11.2. Response

The response is a multi-status response that describes the result of the MOVE-TREE on each effected resource. The response is given for the resource that was to be moved, not the resource that was created as a result of the move. In other words, each entry indicates if the move on the resource specified in the href succeeded or failed and why.

The exception to this rule is for errors that occurred on the destination. For example, if the destination was locked the response would indicate the destination URL and a 416 "Locked" error.

11.3. Example

```
MOVE-TREE  /container/ HTTP/1.1
Host: www.foo.bar
Destination: http://www.foo.bar/othercontainer/
Enforce-Live-Properties: *
Overwrite: False
State-Token: <OpaqueLockToken:xxxx> <OpaqueLockToken:xxxx>


HTTP/1.1 207 Multiresponse
Content-Type: text/xml
Content-Length: xxxxx

<?namespace href = "http://www.ietf.org/standards/dav/" As = "D"?>
<d:multiresponse>
  <d:response>
       <d:href>http://www.foo.bar/container/resource1</d:href>
       <d:href>http://www.foo.bar/container/resource2</d:href>
       <d:href>http://www.foo.bar/container/</d:href>
       <d:href>http://www.foo.bar/container/C2/R2</d:href>
       <d:status>HTTP/1.1 201 Created</d:status>
  </d:response>
  <d:response>
       <d:href>http://www.foo.bar/container/C2</d:href>
       <d:status>HTTP/1.1 418 Method Failure</d:status>
  <d:response>
       <d:href>http://www.foo.bar/othercontainer/C2</d:href>
```

```
          <d:status>HTTP/1.1 416 Locked</d:status>
      </d:response>
   </d:multiresponse>
```

In this example the client has submitted a number of lock tokens with the request. A lock token will need to be submitted for every resource, both source and destination, anywhere in the scope of the method, that is locked. In this case the proper lock token was not submitted for the destination http://www.foo.bar/othercontainer/C2. This means that the resource continer/c2 could not be copied, although its child container/C2/R2 could be copied.

12. 102 "Processing" Response Code

The *-Tree methods can potentially take a long period of time to process. In such cases the client may time-out the connection while waiting for a response. To prevent this the server MAY return a 102 response code to indicate to the client that the server is still processing the method.

If a method is taking longer than [INSERT NUMBER HERE] seconds to process the server SHOULD return a 102 "Processing" response.

13. Status-URI Response Header

The Status-URI response header MAY be used with the 102 "Processing" response code to inform the client as to the status of a method.

Status-URI = "Status-URI" ":" *(Status-Code "<" URI ">") ; Status-Code is defined in 6.1.1 of [RFC2068]

The URIs listed in the header are source resources which have been effected by the outstanding method. The status code indicates the resolution of the method on the identified resource. So, for example, if a COPY-TREE method is outstanding and a 102 "Processing" response with a Status-URI response header is returned, the included URIs will indicate resources that have had copy attempted on them and what the result was. Note that including the URI does not indicate the result of applying the method.

14. Author's Address

Yaron Y. Goland
Saveen Reddy
Microsoft Corporation
1 Microsoft Way
Redmond, WA. 98053
USA

e-mail: {yarong, saveenr}@microsoft.com

15. Bibliography

[Goland et al., 1997] Y. Goland, E. J. Whitehead, Jr., Asad Faizi, Stephen R. Carter, Del Jensen 'Extensions for Distributed Authoring and Versioning on the World Wide Web -- WEBDAV', March 1997, <URL: ftp://ftp.ietf.org/internet-drafts/draft-ietf-webdav-protocol-04.txt>

[RFC2068] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, 'Hypertext Transfer Protocol -- HTTP/1.1', RFC 2068, January 1997, <URL:ftp://ds.internic.net/rfc/rfc2068.txt>