Extensions for Distributed Authoring and Versioning
on the
World Wide Web -- WEBDAV

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

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

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

The HTTP working group at <http-wg@cuckoo.hpl.hp.com> discusses
the HTTP protocol. Discussions of the HTTP working group are
archived at
<URL:http://www.ics.uci.edu/pub/ietf/http/>;
general discussions about HTTP and the applications which use
HTTP should take place on the <www-talk@w3.org> mailing list.

Abstract
WEBDAV (Web Distributed Authoring and Version control) specifies a set of methods and content-types ancillary to HTTP/1.1 for the management of resource meta-data, simple name space manipulation, simple resource locking (collision avoidance) and resource version control.
Abstract .............................. 2
1. Introduction ....................... 7
  1.1 Purpose .......................... 7
  1.2 Terminology ........................ 7
  1.3 Notational Conventions and Generic Grammar ........................... 9
  1.4 Design Conventions .................. 9
2. Links .............................. 11
  2.1 Introduction ....................... 11
  2.2 Link Types .......................... 11
  2.3 LINK ............................. 11
  2.3.1 Method Definition ................ 11
  2.3.2 Request Body ..................... 12
  2.3.3 Response Body .................... 13
  2.3.4 Error Conditions .................. 13
  2.5 UNLINK ............................ 13
  2.5.1 Method Definition ................ 13
  2.5.2 Request Body ..................... 13
  2.5.3 Response Body .................... 14
  2.5.4 Error Conditions .................. 14
  2.6 LINKSEARCH ....................... 14
  2.6.1 Method Definition ................ 14
  2.6.2 Request Body ..................... 14
  2.6.3 Response Body .................... 15
  2.6.4 Error Conditions .................. 16
  2.6.5 Response Codes ................... 16
  2.7 GETLINKS ........................... 17
  2.7.1 Method Definition ................. 17
  2.7.2 Request Body ..................... 17
  2.7.3 Response Body .................... 17
  2.7.4 Response Codes ................... 17
  2.8 GETLINKVAL ....................... 17
  2.8.1 Method Definition ................. 17
  2.8.2 Request Body ..................... 17
  2.8.3 Response Body .................... 17
  2.8.4 Response Codes ................... 18
  2.9 SETLINKVAL ....................... 18
  2.9.1 Method Definition ................. 18
  2.9.2 Request Body ..................... 18
  2.9.3 Response Body .................... 18
  2.9.4 Response Codes ................... 18
3. Distributed Authoring and Versioning Link Schema ........ 18
  3.1 DAV.Versioning.History ............. 19
  3.1.1 History Link Definition ............ 19
  3.1.2 History Link Type ................. 19
  3.1.3 Destination Body .................. 19
4. Locking ............................. 21
  3.2 DAV.Versioning.CheckedOut ............ 20
  3.3 DAV.Versioning.DefaultPublished ....... 20
  3.4 DAV.Versioning.NextVersion ........... 20
  3.5 DAV.SupportedLinkTypes ............. 21
  3.6 DAV.SupportedLinkSchemas .......... 21
  3.7 DAV.Source .......................... 21
1. Introduction

[Editor’s Note: Method parameter and response packaging is still being debated, as is use of Web Collections; the packaging given in this document should not be considered final or settled.]

1.1 Purpose

The purpose of this document is to extend the HTTP 1.1 [HTTP11] protocol to support features which satisfy the Distributed Authoring and Versioning requirements set down in [INSERT REFERENCE TO UNIFIED DAV REQUIREMENTS DRAFT].

1.2 Terminology
The following entries define how a term is used within this document.
Unless otherwise noted, the use of terminology in this document is consistent with the definitions of terms given in [HTTP11].
Editor: Sections references need to be entered.

arbiter
A resource which performs actions on the behalf of other resources.

attribute
An attribute is a name/value pair which contains meta-information regarding a resource. See section x.x.

attribute set
An attribute set is a collection of attributes. See section x.x.

set edit announcement
A set edit announcement is a declaration by a principal that they intend to edit a resource. No action other than the declaration is implied by a set edit announcement. See section x.x.

remove edit announcement
A remove edit announcement is a declaration by a principal that they no longer intend to edit a resource. See section x.x.

check-out root
One or more versioned resources within a particular version tree which are operated on by a CHECKOUT method.

command comments
Command Comments associate either a string or a URI with an HTTP message. See section x.x.

copy
A copy performs a duplication of a resource, making it available at both the original and new location in the URI namespace. Due to contextual differences, it may not be possible to create an exact copy of a resource. A copy only requires best effort on the part of the copy agent. See section x.x.

collection
A group of URIs which are manipulated as a unit. See section x.x.

destroy
To destroy a resource is to request that the resource be permanently removed from storage. This differs from delete in that some versioning systems handle delete as a request to no longer make the specified resource available under the specified URI. See section x.x.

diff
A diff is a mechanism whereby two or more resources are compared and the differences enumerated. See section x.x.

history
A history lists the URIs of the versions of a resource along with related information. See section x.x.

merge

Goland, et al

INTERNET-DRAFT WEBDAV March 26, 1997
A merge is the process whereby information from multiple resources is folded into a single resource. Merges can occur at the client or the server. See section x.x.

move

A move is logically an atomic action consisting of a copy followed by a delete of the source resource. Please see the definition of copy in this section. See section x.x.

no-modify lock

A no-modify lock prevents a locked resource from being edited until all no-modify locks on the resource are released. See section x.x.

notify request

A notify request instructs the recipient to send information regarding the progress of a request. See section x.x.

principal

A principal is the source of a message. For example: persons, computers, and programs.

read lock

A read lock prevents principals who do not possess a read lock on a resource from reading that resource. See section x.x.

redirect

A redirect instructs a server to return one of the 3xx series codes. See section x.x.

relationship

A relationship is a unidirectional typed link. See section x.x.

shared mode

Shared mode modifies a lock request such that the lock may be shared between multiple principals. See section x.x.

version identifier

The identifier used to name a version of a resource.

working resource

The writeable resource created by some versioning engines as the result of a CHECKOUT method invocation.

write lock

A write lock prevents principals who do not possess a write lock on a resource from editing that resource. See section x.x.

1.3 Notational Conventions and Generic Grammar

This specification uses the Augmented BNF and Basic Rules as defined in Section 2 of [HTTP11].

1.4 Design Conventions

The following design conventions have been adopted by the HTTP Distributed Authoring and Versioning (DAV) group.

1. Whenever reasonable new functionality will be expressed as new methods. When sensible the commands for such a method will be included in the message body of the
request.

2. This resolution was adopted in order to solve a debate about how to express new functions such as copy and diff. One group wished to use POST and MIME types to express new commands. Another group wished to use new methods and MIME types to express new commands. A third group wished to use new methods and new method headers to express new commands. The group settled on using new methods and MIME types to express new commands. The new methods would allow for quick parsing by proxies and servers but the MIME types would allow for flexibility in specifying the command. It was also felt that introducing method specific headers would violate the letter and spirit of the HTTP protocol.

3. Methods should only perform a single action.

4. The original design of many of the methods in this document allowed for multiple requests of the same type to be packaged together. So, for example, one could send a Copy method which specified that 1000 different URIs should be copied. It was decided that in the face of pipelining, compressed headers, and other innovations there was no need to add this level of complexity to the protocol. This decision was based upon the assumption that a new method or MIME type would be introduced which would allow for any number of HTTP requests to be bundled together. It was further assumed that such a method or MIME type would carry with it the option to specify that the contents be processed atomically.

5. URLs are opaque tokens and should be treated as such.

6. It is often possible to decrease the number of trips to retrieve a particular piece of data by putting suffixes on URLs. However URLs are intended to be opaque tokens. The above design principal not prevent the use of suffixes but rather requires that the server indicates that a particular URL understands a particular type of suffix.

7. Only addressable entities will be subject to the commands in this document.

8. Response must be machine processable. Currently HTTP responses rely upon humans being available to view the response and take appropriate action. This level of error encoding is not sufficient for WEBDAV purposes where all actions and results of all actions must be fully machine processable. However, a corollary of this
design principal is that response messages are not required to provide information about incorrect syntax, only to point out that incorrect syntax has been submitted. It is legitimate to assume that the client and server are both capable of generating proper syntax. However, this corollary in no way effects the general principal of being generous in the sorts of messages accepted.

It is sometimes desirable to have a non-addressable entity. For example, an accept header may specify a content-language. The response may contain a representation in that language but may not contain a content-location header. In that case the representation of that resource in that language will not alterable with the commands given in this document.

2. Links

2.1 Introduction

In [HTTP11] link and unlink methods were introduced. The idea was to allow for two or more arbitrary resources to be associated

Goland, et al

INTERNET-DRAFT WEBDAV March 26, 1997

with each other. In essence, its purpose was to extend the linking facilities available on the Web to all content types, not just HTML. This document proposes a link facility consisting of links which are typed and unidirectional, containing an explicit source and destination.

2.2 Link Types

Links are typed. These types explain the purpose and behavior of a link. It is possible that multiple links defined on the same resource may have the same type, e.g., a resource with multiple authors.

A link type usually belongs to a link schema, such as the "LockInfo" type which belongs to the "DAV" schema. By convention, links which belong to a schema are specified as: schema "." typename. Thus, the LockInfo type would be specified as "DAV.LockInfo". This creates a type namespace which is less susceptible to namespace collision than a single, flat namespace. However, use of the "." to denote hierarchy is only a convention, since types are properly considered to be opaque tokens.

2.3 LINK

2.3.1 Method Definition

The LINK method requests that a single typed unidirectional hypertext link be created on the Request-URI. This link consists
of three fields, a source URI, a destination URI, and a type, which is an opaque token. The source URI is the beginning of the link, the destination URI is the endpoint of the link, and the type describes the relationship between the source and destination resources. A link may be interpreted as a binary relationship between the source and destination resources, which has the meaning, "the destination resource contains information of relation <type> to the source resource." A link may also be interpreted as an attribute-value pair, where the attribute name is the link type, and the value is pointed to by the destination of the link.

The specification of the requested link is made in the message body of the LINK request, which SHOULD be of content type "application/link". The source URI of the link will typically correspond to the Request-URI of the LINK request, but MUST not be required to do so. Either the source URI or the destination URI of the link MUST correspond to the Request-URI of the LINK request. This ensures that the Request-URI is part of the link being defined.

If the source URI is omitted in the link specification, the source URI defaults to the Request-URI. If the destination URI is omitted in the link specification, the destination URI defaults to the Request-URI.

Some servers may automatically allocate part of their namespace for link information of a specific type. For example, the creation date of a resource at URL <http://www.ics.uci.edu/resource.html> might be accessed from <http://www.ics.uci.edu/resource.html?create_date>. In this case, a user-agent would not want to specify the destination of a link of type "create_date," and instead would want the server to automatically fill-in the correct destination URI. If the server should specify the source or destination URI of a link, then the key word "SET" is entered for the source or destination URI in the link specification.

2.3.2 Request Body

The request body of a LINK method request is of content type application/link, defined as:

```
Link = Version CRLF [ Source CRLF ] [ Destination CRLF ] Type CRLF
Schema-Version = "Schema-Version" ":" 1*Digit "." 1*Digit
Source = "Source" ":" ( URI | "SET" )
Destination = "Destination" ":" ( URI | "SET" )
Type = "Type" ":" token
```

The Schema-Version field specifies the version of the specification of the application/link media type. By definition, the version of the application/link media type specified in this document is "1.0".

Example
2.3.3 Response Body

The request body of a LINK method request is of content type application/linkresult, defined as:

LinkResult = Version CRLF LinkSource CRLF LinkDestination CRLF Type CRLF

Goland, et al
application/unlink, defined as:

UnLink = Version CRLF Source CRLF Destination CRLF Type CRLF
Schema-Version = "Schema-Version" ":" 1*Digit "." 1*Digit
LinkSource = "Source" ":" URI
LinkDestination = "Destination" ":" URI
Type = "Type" ":" token

The Schema-Version field specifies the version of the specification of the application/unlink media type. By definition, the Schema-Version of the application/unlink media type specified in this document is "1.0".

2.5.3 Response Body

The response body of an UNLINK method is of content type application/unlinkresult, defined as:

UnLinkResult = Version CRLF 1*(LinkSource CRLF LinkDestination CRLF Type CRLF)
Schema-Version = "Schema-Version" ":" 1*Digit "." 1*Digit
LinkSource = "Source" ":" URI
LinkDestination = "Destination" ":" URI
Type = "Type" ":" token

2.5.4 Error Conditions

a) there is a syntax error in the application/unlink message body (e.g., missing fields, missing field values, missing type value)
b) Both the source URI and the destination URI are specified, and neither match the Request-URI.
c) No links match the specification in the request message body.

2.6 LINKSEARCH

2.6.1 Method Definition

The LINKSEARCH method is used to search on the link tuple space. The search occurs in the URI namespace. Search points and scope define the area of name space to be searched. The search points are URIs from which the search should start. The scope defines the number of levels into the URI namespace the search should go down. The search syntax is in prefix notation in order to ease processing by the server.

2.6.2 Request Body

The Request-URI is the search arbiter, and it is the job of that resource to process the request and return the results. The media type is application/linksearch.
This syntax is adapted from [RFC1959] and [RFC1960].

LinkSearchMethod = Schema-Version CRLF [SearchPoints] CRLF
SearchPattern CRLF Scope CRLF
SearchPoints = "SearchPoints" ":" 1#URI
SearchPattern = "SearchPattern" ":" 1*SearchOperations
SearchOperations = And | Or | Not | Item
And = "&" 2*SearchOperations
Or = "|" 2*SearchOperations
Not = "!" SearchOperations
Item = Operator CompareTo MatchPattern
Operator = "=" | "!="

The SearchPoints are URIs on which the search is to be performed. If at least one URI is not listed in SearchPoints then the search point becomes the request-URI.

The operator is equals "=" or not equals "!=". Source and Destination equality are dependent upon the type of URIs. Please refer to the definition of the referenced URI type to determine equality rules. The server MUST NOT compare URIs with which it is not familiar. Two types are equal if their tokens are octet for octet equal.

The MatchPattern allows for wildcard matching. Inserting spaces prevents the wildcard character to not be confused with characters in the token. A wildcard character matches zero or more characters.

The scope operator specifies how deep into the URI namespace the search should go. A scope of 0 means the search is only on the search point(s). A scope higher than zero means the search should proceed scope levels down the URI namespace. A scope of "sub" means the search should go all the way down to the leaves of the namespace.

Redirects may only be followed if the new location is within the defined search space.

2.6.3 Response Body

The response body will be of type text/html and contain a web collection with type "DAV/LinkSearchResponse". That web collection will itself contain a series of web collections that have the following definition:

WC Attributes

Type = "DAV/LinkResponse"
WCDATA Entry

Entry = [OnSource] [OnDestination] Type
OnSource = ["ON"] "Source" ":" URI CRLF
OnDestination = ["ON"] "Destination" ":" URI CRLF
Type = "Type" ":" token CRLF

Each DAV/LinkResponse indicates the source, destination, and type

Goland, et al

INTERNET-DRAFT WEBDAV March 26, 1997

of a link result. The "ON" tag indicates if the source and/or destination is aware of the link’s existence.

2.6.4 Error Conditions

Editor: these error conditions will be merged with the response codes in the following section in a future rev. of the document.

URI Doesn’t Exist (From Here on Down) (Incomplete) -

URI Access is Forbidden (From Here on Down) (Incomplete) - 403 Forbidden

URI Access is Not Allowed (From Here on Down) (Incomplete) - 401 Unauthorized

URI Not Found (From Here on Down) (Incomplete) - 404 Not Found

Arbiter does not Support Search on this URI (From Here on Down) (Incomplete) - 406 Not Acceptable

URI does not Support Search (From Here on Down) (Incomplete) - 405 Method Not Allowed

URI has been redirected out of search space (Temporarily | Permanently)(From Here on Down)(Incomplete) - 301 Moved Permanently, 302 Moved Temporarily

Unknown URI Type in Comparison (Include URI)

2.6.5 Response Codes

202 Accepted - The request has been accepted and an appropriate response generated.

203 Non-Authoritative Information - The request has been accepted but the response may be based on outdated or non-authoritative link information.

413 Request Entity Too Large - The results of the request are larger than the server is willing to return. The request should be made again using an appropriate range header. No body is returned.
2.7 GETLINKS

2.7.1 Method Definition

GETLINKS is a convenience method that takes a type and returns a webmap containing link tuples whose source or destination equals the Request-URI and whose type matches the submitted typed.

2.7.2 Request Body

The media type of the request body is application/GETLINKS.

GetLinksMethod = Schema-Version CRLF TaggedType CRLF
TaggedType = "Type" ":" #type | "All"

2.7.3 Response Body

A text/html media type containing a web collection of type DAV/GetLinksResponse that contains DAV/LinkResponse entries.

2.7.4 Response Codes

See the return codes for the LinkSearch method.

2.8 GETLINKVAL

2.8.1 Method Definition

GetLinkVal is a convenience method that takes a type and returns a message/multipart. The first entry in the message/multipart is a Web Collection with link tuples whose source or destination equal the Request-URI and whose type matches the submitted type. The rest of the entries in the message/multipart are the result of a GET submitted on the source or destination of the link that does not equal the Request-URI.

2.8.2 Request Body

The request body for method GETLINKVAL is of content type application/getlinkval, defined below:

GetLinkValMethod = Schema-Version CRLF TaggedType CRLF

2.8.3 Response Body

The response body is multipart/related. The first entry is of type text/html containing a DAV/GetLinkValResponse Web Collection containing DAV/LinkResponse web collections. Following the text/html is a series of responses containing GET results.
2.9 SETLINKVAL

2.9.1 Method Definition

SETLINKVAL is a convenience method that combines LINK and PUT. It is used to create a link and to specify the contents of one end of the link.

2.9.2 Request Body

The request body consists of a MIME multipart. The first entry is the request body contained below which is of type Application/SETLINKVAL. The second entry is the headers and body of the information to be PUT.

```
SETLINKVAL = Schema-Version CRLF LINK CRLF PutDestination CRLF ;
```

```
LINK in Section 2.3.2
```

```
PutWhere = "Put-Where" "Source" | "Destination"
```

2.9.3 Response Body

A web collection that contains the results from LINK and PUT.

2.9.4 Response Codes

TBD.

3. Distributed Authoring and Versioning Link Schema

This section specifies only those link types which are absolutely necessary to implement the functionality described in this specification. Link types other than the ones described within are possible, but must be defined in other documents. Since there are many different possible relationships which can exist between resources, there are many different link types which can exist to describe these relationships. It is difficult, if not impossible to enumerate and describe all of these relationships, especially since the meaning of the same type may vary depending on usage context. This specification has therefore chosen not to try and enumerate all possible link types, instead providing a mechanism for discovering which link types (DAV.SupportedLinkTypes), and which sets of link types (DAV.SupportedLinkSchemas) are supported on a resource.

A server is not required to support all of the DAV Link Types.
However, if a server supports a DAV Link Type, it SHOULD maintain the consistency of that relationship by ensuring the contents of the Destination URI are always up-to-date.

The DAV Link Types specified in this document form the "DAV" link schema.

All DAV schema links have a source URI which is the resource being described by the link. The destination URI alway points to a resource whose content type, and contents are described below for each DAV link type.

3.1 DAV.Versioning.History

3.1.1 History Link Definition

The history link returns information about the history of a versioned document. This information is contained within a DAG that describes the ancestors and children of various versions in the tree. The history link MUST be defined on the tree handle and SHOULD be defined on individual versions. Versions that do not have a history link MUST have a tree link defined on them.

3.1.2 History Link Type

The history link has type DAV.Versioning.History.

3.1.3 Destination Body

The destination body of the history link is a text/html file containing a series of web collections of type "DAV/History".

WC Tag Attributes:
Type = "DAV/History"
Name = VersionToken
WCAT Tag Attributes and/or WCDATA Entry:
Rel = ("ParentEntry" | "ChildEntry" | "SiblingEntry")
Version = Token
HREF = Pointer to a web collection w/more information
Rel = ("Parent" | "Child" | "Sibling")
Version = Token
HREF = Pointer to a parent, child, or sibling.
The Rel/Version/HREF triples above can be used with WCAT or WCDATA. The *Entry links provide pointers to web collections with more information about the family member. The version tag provides the version of the relative being pointed to. The second Rel/Version/HREF triple provides a pointer directly to the
versioned resource. All attributes defined on HCs MAY also be used here to provide information such as creation date, author, owner, etc.

3.2 DAV.Versioning.CheckedOut

A server MAY create a link of type DAV.Versioning.CheckedOut on the check out root(s), the working resource, or both. The destination resource of a DAV.Versioning.CheckedOut link is of content type application/checkoutresult, which contains the same information as the response body to the CHECKOUT method invocation which caused the creation of this link. An optional Web Collection attribute, "Principal" may also be included, which holds information about the principal(s) who initiated the check out.

Principal

WC Attributes:

Type = "DAV/Identifier"

WCDATA Attributes:

ContactURI = "URI"

3.3 DAV.Versioning.DefaultPublished.

A server MAY create a link of type DAV.Versioning.DefaultPublished on the version tree handle and all members of the version tree pointed to by the handle. The destination resource of a DAV.Versioning.DefaultPublished link is a member of the version tree, and points to the version which has been designated as being appropriate for display to non-editors. The DAV.Versioning.DefaultPublished destination resource SHOULD be returned to non-DAV aware clients who perform a GET on a version tree handle.

3.4 DAV.Versioning.NextVersion


3.5 DAV.SupportedLinkTypes

Destination resource content type: Web Collection
The contents of the destination resource is a list of the link types which the server will definitely allow to be created on the source URI. The server may allow the creation of links with types other than those contained in the destination resource, but this is not guaranteed.

If a server supports a particular schema, then all link types in the supported schema SHOULD be returned in the destination resource of a SupportedLinkTypes link.

If a server implements the LINK method, it MUST also implement the SupportedLinkTypes link.

3.6 DAV.SupportedLinkSchemas

Destination resource content type: Web Collection
Syntax: TBD

The contents of the destination resource is a list of link schemas. If a link schema is listed, it means that the server SHOULD allow the creation of links with types belonging to that schema. The server may allow the creation of links with types which do not belong to a schema listed in the destination resource, but this is not guaranteed.

If a server implements the LINK method, it SHOULD also implement the SupportedLinkSchemas link.

3.7 DAV.Source

Destination resource content type: varies
Syntax: TBD

The destination URI of a Source link is a location where a user-agent may request the raw data for the source URI, without any source processing by the server (e.g., server-side includes, CGI processing).

4. Locking

4.1 Introduction

The mechanism for locking and unlocking a resource involves the use of a "lock token" (LockToken) which is used to reference the instantiation of the lock. Each lock token instantiates the
occurrence of a lock on one resource or a specified portion of a resource (i.e., a lock on a range of a resource). The locking of multiple resources will cause the instantiation of an equal number of lock tokens.

Lock tokens represent a lock that is specific to the context of a server and therefore cannot be transferred to another server. Nor is it allowable to transfer the LockToken to another "client space" even if both are instantiated on the same server. A "client space" is defined as the data space in the context of the client obtaining the lock. More than one thread can be active in a given "client space."

A mechanism is provided to ensure that a method is executed only if a lock is in place. This mechanism allows a lock header to be included in the method request. The method will be executed only if the LockToken is still valid. Multiple LockTokens may be include in the lock header. Lock headers may reference any LockToken allowing the submission of a method on a resource with a LockToken that refers to another resource.

A lock may designate "write exclusive" on a given resource. "Write exclusive" is currently the only type of lock supported. A single range for the "write exclusive" lock may be defined and overlapping ranges are never allowed. Thus a lock on the entire resource would be exclusive of any and all range locks on the same resource. Multiple range locks may be instantiated for a given resource (even among multiple "user spaces"), but overlapping ranges are not permitted. The following table describes the allowable states:

<table>
<thead>
<tr>
<th>Request</th>
<th>State</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>full</td>
<td>full</td>
<td>denied</td>
</tr>
<tr>
<td>full</td>
<td>partial</td>
<td>denied</td>
</tr>
<tr>
<td>full</td>
<td>none</td>
<td>granted</td>
</tr>
<tr>
<td>partial</td>
<td>full</td>
<td>denied</td>
</tr>
<tr>
<td>partial</td>
<td>partial</td>
<td>granted*</td>
</tr>
<tr>
<td>partial</td>
<td>none</td>
<td>granted</td>
</tr>
</tbody>
</table>

* if no overlap

Once defined the definition of the lock range may not be modified. The only mechanism supported is to release the lock and instantiate a new lock with a defined range.

The size of a resource may not be changed within the defined region of a lock range unless the range includes +n (n>0) octets beyond the end of the resource (eof). By definition, locking a range beyond the eof provides the owner with the ability to add to the eof and therefore extend the resource.
If an entire resource is locked and the lock owner deletes the resource then the write lock remains. Further actions on the resource (e.g. PUT) are allowed because the LockToken is still instantiated.

Each lock may have a time out associated with it. If no time out value is associated with the lock then the lock will never time out (implementors should have some mechanism for destroying a lock at an administrative level). Otherwise the lock will expire if the designated amount of time has elapsed without a method being invoked on the resource by the lock owner. The time out value may be modified without affecting the rest of the the lock by submitting a lock request for the LockToken with no other values associated with it other than a new time out value.

4.2 LOCK

4.2.1 Method Definition

[Ed. Note: Need description of LOCK method.]

The Request-URI of the LOCK method is the URI of the resource to be locked.

4.2.2 Request Body

The lock request specifies the target of the lock, parameters of the lock, and requests that a lock be set on the resource.

The request body of a LOCK method is of content type application/lock, defined as:

```
LockBody = Schema-Version LockEntry CRLF TimeOut CRLF
          [LockRange CRLF] [LockOwner CRLF]
LockEntry = "LockEntry" ":" LockWriteExclusive
LockWriteExclusive = "write exclusive"
TimeOut = "TimeOutSeconds" ":" TimeOutVal
TimeOutVal = 1*digit | "Infinite"
LockRange = "LockRange" ":" 1*digit "-" (1*digit | "eof++")
LockOwner = "LockOwner" ":" URI

Example:
LOCK /users/johndoe/davspec.html HTTP/1.1
Content-Type: application/lock
Schema-Version: 1.0
LockEntry: write exclusive
TimeOut: 2400
LockRange: 100-eof++
LockOwner: mailto:johndoe@foo.com
```
The response body of a LOCK method is of content type
application/LockResponse, defined below. The syntax of this
content type is that of a Web Collection.

ResourceLockResponse = WC_Open LockResponse WC_Close
WC_Open = "<"WC VERSION="1.0" TYPE="
("DAV/Lock"|"DAV/Unlock")
"">NAME=""> LockToken "">=">
LockToken = token
LockResponse = *WCAT LockData *WCAT
LockData = "<WCDATA> LockURI LockBody LockGranted "</WCDATA>"
; LockBody production from Section 4.2.2
LockURI = "ResourceLocked" ":" URI
LockGranted = "LockGranted" ":" RFC1123-date [*digit] ;*digit
for milliseconds
WC_Close = "</WC>"
Example:

This is a possible response body from the lock request example in
Section 4.2.2.

<WC VERSION="1.0" TYPE="DAV/Lock" NAME="LK_Xj5N75">
<WCDATA>
Schema-Version: 1.0
LockEntry: write exclusive
TimeOut: 2400
LockRange: 100–eof++
LockOwner: mailto:johndoe@foo.com
LockGranted: Thu, 23 Jan 1997 15:11:27 GMT
</WCDATA>
</WC>

4.2.4 Error Conditions

Lock already owned:
Lock range overlaps:
Invalid range:
Resource not found:
Denied:
General Failure:

Goland, et al. 24
resource to be unlocked.

4.3.2 Request Body

The request body of an UNLOCK method is of content type application/unlock, defined below:

UnlockRequest = Schema-Version UnLockToken
UnLockToken = "UnLock-Token" :: LockToken

4.3.3 Response Body

The response body of an UNLOCK method is of content type application/UnlockResult, which is defined by the production ResourceLockResponse, defined in Section 4.2.3.

4.3.4 Error Conditions

Lock not found:
Lock not owned:
Lock timed out:
Denied:
General Failure:

4.4 Lock Discovery

[Ed. Note: "LockInfo" link and associated "LockDiscovery" WC should be included in the section on DAV Link Schema links]

4.4.1 Lock Discovery via GetLinkVal

When a lock is taken out the system SHOULD record who owns the lock. Ownership information can be taken from the From header, from identification provided by authentication, or from the LockOwners field.

Lock Discovery is enabled via a DAV Schema link of type DAV.LockInfo. Lock Discovery is performed by using the GetLinkVal method on links of this type.

Goland, et al

INTERNET-DRAFT WEBDAV March 26, 1997

The LockInfo Link is structured as:

src = URI of the resource
dest = URI of the lock discovery WC
type = "LockInfo"

4.4.2 Lock Discovery Destination Resource

The DAV.LockInfo destination resource is of content type text/html, which uses the Web Collection syntax:

LockDiscoveryResponse = WC_Open LockResponse OtherFields WC_Close
5. Name Space Manipulation

5.1 COPY

5.1.1 Method Definition

This method requests the server to manipulate the mapping for the resource in the namespace such that a new location (mapping) is created for the Request-URI in the namespace, while maintaining the existing location (mapping) in the namespace. The actual implementation is left to the discretion of the server. The presence of hyperlinks within a resource creates special problem in Web resources. The fact that these hyperlinks could be either absolute or relative create yet another level of complexity.

The Request-URL will be the URL for the source resource which will be copied to the destination. In case of a versioned resource, depending on the implementation, it may be either the root of the version tree, in which case it may mean to copy the whole tree or a specific version (depending on the value of request-version), or it may be the version tree history. In case of non-versioned resource, it will refer to the resource itself.

The message body will contain the URL of the destination of the COPY. The OVERWRITE header, as defined by this spec., defines the behavior of the server if a resource already exists as referred to by destination URL. If the value of the overwrite header is true (default), then any existing resource referenced by the destination URL will be overwritten. If the value of overwrite header is false then the copy will fail when the URL of an existing resource matches that of the destination URL. In case of non-versioned collections, COPY can be used as one of the ways to add to the collection.

5.1.2 Request Body

The information about the destination and the switch to determine if the links need to copied as well, will be passed in the
message body which should be of the content type "application/copy". Also defined in the body is a switch called "copylink". If the value of this switch is set to true, then the server should copy all the links that it deems pertinent. This spec. leaves it to the discretion of the server to determine which links are or are not appropriate in the context of copy. When links are copied the following condition MUST be met:

The types of the links should be maintained across the copy.

If the source field of a link matches Request-URI, then the source field of the link on the copy destination should be destination URI.

The definition of the content type Application/Copy is as follows:

copy = Schema-Version CRLF destination CRLF copylink CRLF
[request-version CRLF]
Schema-Version = 1*DIGIT "." 1*DIGIT
destination = "destination" ":" URI
copylink = "copylink" ":" value
value = "true" | "false"
request-version = "Request-Version" ":" token

Note: COPY and Version Trees

The request-version token may be used to refer to a versioned resource by its version ID, in which case the Request-URI must point to the version tree handle. If the Request-URI points to the version tree handle and the request-version field is not provided, then the whole tree will be copied to the new location.

The following rules apply when a versioned resource is involved in a copy operation:

A versioned resource can be copied into a non-versioned space.

A versioned resource cannot be copied into another versioned resource.

A non-versioned resource cannot be copied into a versioned resource.

5.1.3 Response Body

The response body returns the status code back.

5.1.4 Error Conditions

The following status codes may be returned:
OK - The server has successfully copied the resource, and all the links, if applicable, to the new location in the namespace.

Copy failed due to some internal error:

- The source is missing
- The destination exists (and the overwrite switch is false)
- One or more links (in case copylink switch is true) failed to copy due to some internal error.

5.2 MOVE

5.2.1 Method Definition
This method requests the server to manipulate the resource/namespace mapping; a new location (mapping) is created for the Request-URI in the namespace, and the existing location (mapping) in the namespace is removed. The actual implementation is left to the discretion of the server.

Move can be considered as an atomic, logical operation combining a COPY followed by a DELETE on the source URI. This means MOVE inherits all the complexities of COPY operation, and add a few more of its own due to the fact that it changes the resource location in the namespace.

For instance: from the perspective of the client, a MOVE on a resource may result in a REDIRECT to the new location namespace. WEBDAV does not require a server to issue a redirect on "moved" URIs in order to be compliant, but one might argue that some sort of discovery for "server capability" be provided.

Also there is an issue of "level of intelligence" of move. One argument is that due to the special nature of HTML media types, move should automatically perform some link management on links after a MOVE. While it is impossible to manage all links pointing to a resource due to lack of control over all of the links pointing to the resource, it is conceivable that links from a resource, and links within a resource might be manageable. The opposing argument to this is that allowing a move operation to have side-effects which could modify the resource being moved - or worse yet modify resources which were not being moved - can have unexpected consequences. It is important to know the consequences of an operation before executing the operation. WEBDAV is silent on MOVE/link interaction issues, specifying a simple MOVE with a discoverable list of links to be moved with the resource. Beyond that WEBDAV does not make any recommendation or suggestions about the actual implementation of the MOVE operation.
In case of non-versioned collections, move can be used as one of the ways to add to the collection.

5.2.2 Request Body

This section defines the content type application/move.
Move = Schema-Version CRLF destination CRLF movelink CRLF [request-versionCRLF]
Schema-Version = 1*DIGIT "." 1*DIGIT
destination = "destination" ":" URL
movelink = "movelink" ":" value
value = "true" | "false"
request-version = "Request-Version" ":" token

Note: COPY and Version Trees

The request-version token may be used to refer to a versioned resource by its version ID, in which case the Request-URI must point to the version tree handle. If the Request-URI points to the version tree handle and the request-version field is not provided, then the whole tree will be moved to the new location.

The following are the rules that apply when a versioned resource is involved in a move operation:

A versioned resource can be moved into a non-versioned space.

A versioned resource cannot be moved into another versioned resource.

A non-versioned resource cannot be moved into a versioned resource.

5.2.3 Response Body

The response body returns the status code.

5.2.4 Error Conditions

The following status codes may be returned:

OK - The server has successfully moved the resource, and all the links, if applicable, to the new location in the namespace.

Move failed due to some internal error:

The source is missing

The desitination exists (in case overwrite switch is false)

One or more links (in case movelink switch is true) failed to
move due to some internal error.

6. URI Collections

6.1 URI Collection Behavior

According to The Random House College Dictionary - Revised Edition, Collection is defined as "2. Something that is collected, as a group of objects..." In the most generic sense a URI Collection brings together a group of URIs and provides information about them. This specification will not be defining such a collection, mostly because it is a pretty idea with nowhere to go. At least for now. In the future systems will be available that will be able to make profitable use of a generic collection. For now this specification defines the Hierarchical Collection (HC). The HC is meant to walk, talk, and look just like a directory in a file system. URIs are added to an HC as either relative or absolute. All relative URIs in an HC are relative to the Request-URI. This restriction is in place in order to replicate normal file system behavior. Thus if the collection is copy, moved, or otherwise manipulated within the name space it will be possible to copy, move, etc. its component parts. This behavior is referred to as propagation. The idea is that a command executed on the collection can be propagated to all the relative URIs. The PropagateLevel header controls propagation behavior. This header specifies the number of levels a command should be propagated. Absolute URIs are added as links and thus commands are not propagated through them. Thus, if a collection is copied, the COPY method will be executed on all the relative URIs but not on the absolute URIs. Of course all the absolute URI entries will still be members of the collection, they simply won’t have the COPY method executed on them.

Executing the MAKECOLLECTION method creates HCs. This is the same as executing a MKDIR on a file system. If a MAKECOLLECTION is executed on a Request-URI which is not itself a member of a collection then either the system will automatically create all the necessary collections or it will reject the method. Once the collection is created any URI PUT under the collection’s namespace will "magically" appear inside the collection. Again, this behavior mimics that of a normal file system. Adding a relative URI directly to a collection will either be rejected or cause a NULL file to appear. Removing a URI directly from a collection will cause a link to disappear in the case of an absolute URI or will cause the file to be deleted in the case of a relative URI.

6.2 CREATECOLLECTION Method
6.2.1 Method Definition

The CREATECOLLECTION method indicates to the server that the clients wishes a specified URI to act as a collection.

6.2.2 Request Body

The contents of message/collection identify what sort of collection is being created. This document defines the Hierarchical Collection (HC). A HC only supports a hierarchical name space and is meant to replicate the behavior of a file system directory.

A HC contains URIs. URIs may be entered as relative or absolute. Relative URIs must be relative to the request-URI. A relative URI is a propagate entry. If a PropagateLevel header is included with a method executed on a HC then the method will be propagated through all relative URIs. Absolute URI entries do not propagate.

MessageCollection = Version CRLF CollectionType CRLF
CollectionType = "Collection Type" "":" ("HC" | Text)

Goland, et al

6.2.3 Response Body

CREATECOLLECTIONERROR = WC_OPEN Type WC_CLOSE
ContentError = "DAV/Error/CreateCollection"
Type = "Do Not Support Type" "":" ("HC" | Text)

6.2.4 Response Codes

6.3 ADDRESOURCE

Editor: this section needs to incorporate the correct text from Yaron.

6.3.1 Content Type Definition

This content type is used to add a link to a collection. In the case of an HC most systems will not accept a relative URI. Rather, to add a new entry to an HC, the resource should be created using a PUT or similar method. If the server does accept a relative URI for an HC then the result SHOULD be the creation of an empty resource.

6.3.2 Request Body

AddResource = Version CRLF URIType CRLF URIEntry CRLF
URIType = "Type" "":" ("Relative" | "Absolute")
URIEntry = "URI" "":" (relativeURI | absoluteURI)
URIType indicates the type of URI while URIEntry indicates the actual URI. URIType is included for negotiation purposes. If the URI is relative then it is relative to the request-URI. The productions relativeURI and absoluteURI are defined in section 3.2.1 of rfc2068.

6.3.3 Response Body

The content type of the response body is text/HTML and contains a Web collection.

6.4 REMOVERESOURCE

6.4.1 Method Definition

This content type is used to remove a URI from a collection. In the case of an HC most systems will not accept a relative URI.

Goland, et al

INTERNET-DRAFT WEBDAV March 26, 1997

Rather, to delete an HC entry, the resource should be DELETEd or DESTROYed. If the server does accept a relative URI for an HC then the result SHOULD be the deletion of the resource.

6.4.2 Request Body

RemoveResource = AddResource

6.4.3 Response Body

The content type of the response body is text/HTML and contains a Web collection.

6.5 PropagateLevel Header

6.5.1 Header Definition

The PropagateLevel header controls how many levels a method will be propagated down a collection. Zero means the method should only be executed on the collection. Infinity means the method should be propagated as far down as possible. A loop is impossible with HCs because its name space is a tree and absolute URIs do not propagate. Other collection types will have to define their own mechanisms to detect and prevent loops.

6.5.2 Request Body

PropagateLevel = "PropagateLevel" ":" (1*Digit | "Infinity")

6.5.3 Response Body
The following web collection can be included in any web collection based body.

WC Attributes

Type = "DAV/Error/Prop/FailedAt"

WCDATA Entry

Entry = "Propagation Mechanism Failed at" "(:" *(Relative | Absolute_URI CRLF)
Relative = "Base" "." Absolute_URI "Relative" "." *Relative_URI CRLF

7. Version Control

7.1 Introduction

This specification provides interface definitions which clients may use to perform the common versioning operations check in, check out, version history retrieval, and merge. A fundamental assumption of this specification is that the semantics of versioning operations are defined by the server, and may vary from server to server, depending on the versioning engine employed. As a consequence, this specification provides a common interface for accessing the functionality of many different versioning engines.

Since versioning functionality may vary from server to server, or even in different namespaces on the same server, a discovery mechanism is provided for clients to determine precisely what effect a CHECKOUT or a CHECKIN method invocation will have on the state of the server. A client may query the server for its CHECKIN and CHECKOUT semantics by retrieving the destination resource of the DAV.Discovery.CHECKOUT and DAV.Discovery.CHECKIN links. It is expected that a client will query the value of the DAV.Discovery.CHECKOUT link before performing a check out, and the DAV.Discovery.CHECKIN link before performing a check in, so it knows exactly what will occur when it issues the CHECKOUT or CHECKIN request.

Similarly, the who and when of setting a version identifier varies significantly:

1. A version identifier MAY be specified by the client upon check out.

2. Upon check out, the server MAY accept the client version identifier, or the server MAY assign a version identifier, or the server MAY not assign any version
identifier.

3. If the version identifier has not already been set, the client MAY specify a version identifier upon check in.

4. Upon check in, the server MAY accept the client version identifier, otherwise the server MUST assign a version identifier unless one has already been assigned (e.g., during check out).

7.2 Versioning Data Model

A version tree is a directed acyclic graph of versioned resources. Arcs in the version graph indicate "is-version-of" (or "is-successor-of") relationships. Each versioned resource is a first-class resource, addressable by a URI. A version tree MUST have a handle resource, known as the version tree handle, which is used to refer to the version tree as a whole. The contents of the handle resource MUST be a single URI (a pointer), which is used by the versioning engine to identify the version tree as a whole. The version handle resource MAY point to either the root object of the version tree, or the history resource for the version tree.

The version tree handle MUST have server generated links of type DAV.Versioning.DefaultPublished, DAV.Discovery.CHECKIN, and DAV.Discovery.CHECKOUT, and MAY have a server generated link of type History. Each versioned resource MAY have server generated links of the following types: DAV.Versioning.History, DAV.Versioning.DefaultPublished, DAV.Versioning.NextVersion, DAV.Discovery.CHECKOUT, DAV.Discovery.CHECKIN.

The destination resource of the DAV.Versioning.History link SHOULD describe the version tree, and contain comment information.

The destination of the DAV.Versioning.DefaultPublished link, if this link exists, points to the version which should be returned by default if a GET is performed on the version tree handle URI by a DAV unaware client.

The destination URI of the each DAV.Versioning.NextVersion link, if any such links exist, MUST point to a successor resource in the version tree.

The destination resource of the DAV.Discovery.CHECKOUT link, if this link exists, must contain information describing the capabilities of the CHECKOUT method as implemented by the server on the source resource.

The destination resource of the DAV.Discovery.CHECKIN link, if this link exists, must contain information describing the
capabilities of the CHECKIN method as implemented by the server on the source resource.

Versioning is an orthogonal issue to content negotiation. Each particular variant of a resource MAY be independently versioned, and MAY have its own version tree.

TBD: A Figure showing a sample version tree (using text characters)

Goland, et al

INTERNET-DRAFT WEBDAV March 26, 1997

7.3 CHECKOUT

7.3.1 Method Definition

A CHECKOUT is a declaration by a client that it might edit a versioned resource. The CHECKOUT method is intended to map to the notion of check out supported by the server’s versioning engine. The Request-URI of the check out method is a version tree handle, which identifies the version tree on which the CHECKOUT will be performed.

A server MAY create a working resource as the result of a check out. If the server creates a working resource, it MUST return the URI of this resource in the response body.

All of the activity which occurs during CHECKOUT method invocation MUST be performed atomically.

7.3.2 Checkout Capability Discovery

Editor: this section will be subsumed by a more general method discovery mechanism.

7.3.3 Request Body

The request body of the CHECKOUT method is of content type application/checkout, defined as:

```
Checkout = Schema-Version CRLF [Derived-From CRLF]
[Request-Version CRLF]
[Request-Lock CRLF] [Request-Intent CRLF]
[Request-Working-Loc CRLF] [Request-Visibility CRLF]
Schema-Version = "Schema-Version" ":" 1*DIGIT "." 1*DIGIT
Derived-From = "Derived-From" ":" #token
Request-Version = "Request-Version" ":" token
Request-Lock = "Request-Lock" ":" "Exclusive" "Write"
```
Request-Intent = "Request-Intent" :: ("true" | "false")

Request-Working-Loc = "Request-Working-Loc" :: ("Client" | "Server")

Request-Visibility = "Request-Visibility" :: Visible-When

Goland, et al

INTERNET-DRAFT WEBDAV March 26, 1997

[Visible-Where]
Visible-When = ("Always" | "If_Locked" | "Never")
Visible-Where = ("On_CheckOut_Root" | "On_Working" | "On_Both")

The Schema-Version is the version number of the syntax specification of the application/checkout content type.

Derived-From specifies the version identifier of the check-out root(s) from which the working resource (if it exists) is derived. If more than one version identifier is specified in the Derived-From value, the working resource MAY be the result of a merge of the specified versions, and the exact semantics of this merge operation will vary across versioning engines.

Request-Version specifies a client’s request to set the version identifier of the working resource. The server MAY set the version identifier of the working resource to this value. If this field is omitted from the request message body, the server MAY perform a check out from a default check out root (often the latest version).

Request-Lock is used by the client to request a lock when a server supports optional locking on check out.

Request-Intent is used by the client to request the server to establish an intent to edit on check out. An intent to edit MAY result in the association of authentication or user information with either the check out root, or the working resource.

Request-Working-Loc is used by the client to request where the working resource is located. If the client specifies "Client," then the server SHOULD not create a working resource on the server. If the server does not create a working resource on the server, the client SHOULD retrieve its working resource from the check out root after successful completion of the CHECKOUT. If the client specified "Server," then the server SHOULD create a working resource on the server.

Request-Visibility is used by the client to request whether to create a DAV.Versioning.CheckedOut link, and the resource which will store the link. Visible-When describes whether the DAV.Versioning.CheckedOut link SHOULD always be created ("Always"), SHOULD only be created if the check out performs a lock ("If_Locked"), or SHOULD never be created ("Never"). Visible-Where describes where the link SHOULD be stored: on the
check out root ("On_CheckOut_Root"), on the working resource ("On_Working"), or on both the check out root and the working resource.

Goland, et al

INTERNET-DRAFT WEBDAV March 26, 1997

7.3.4 Response Body

The response body of a CHECKOUT method invocation is of content type application/checkoutresult, defined as:

CheckoutResult = WC-Open CheckoutResultData WC-Close

WC-Open = "<WC VERSION ="1.0" TYPE ="DAV/CheckoutResult">"

CheckoutResultData = *WCAT WCCheckoutData WCAT

WCAT = {arbitrary, server-defined Web Collection attributes}

WCCheckoutData = "<WCDATA>" DerivedTree DerivedRoot [WorkURI] [WorkVersion] [Visibility]

Derived-Tree is the location of the version tree handle on which the check out was performed.

Derived-Root is the location of the (possibly many) versioned resources on which the check out operation was performed.

WorkURI is the location of the working resource generated by the server on check out, if it exists.

WorkVersion is the version identifier of the working resource, if it was set on check out.

Visibility is the location where the client can retrieve the DAV.Versioning.CheckedOut link.
7.3.5 Error conditions

An error condition occurs if the following conditions hold:

If a client requests a version identifier on check out, but the server:
   a) cannot set a version identifier
   b) refuses to set a version identifier

If a client requests a lock, but the server doesn’t support locking.

If the client requests an intent to edit, but the server doesn’t support intent to edit.

If the client doesn’t specify a Derived-From version, and the server has no default value.

Syntax error in the request body. If a client requests a feature the server doesn’t support then it has committed a syntax error because the discovery told the client what the server supports.

7.4 CHECKIN

7.4.1 Method Definition

A CHECKIN is a declaration by a client that it wishes to place a resource under version control. The CHECKIN method is intended to map to the notion of check in supported by the versioning engine. The Request-URI for a CHECKIN method invocation is the version tree handle of the version tree which is being operated on.

A check in SHOULD clear any server state which is set as a result of a check out. For example, if a server locks a resource on check out, it should remove the lock upon check in.

To place a resource under version control for the first time, a check in is performed on the resource. In this case, the Request-URI is the desired location of the version tree handle, and the Working-URI or the check in body gives the contents of the resource to be placed under version control.

All of the activity which occurs during CHECKIN method invocation MUST be performed atomically.
7.4.2 Checkin Capability Discovery

Editor: this section will be subsumed by a more general purpose discovery mechanism

7.4.3 Request-Body

The request body of a CHECKIN method invocation MUST be in one of two forms:

1. An entity of content type application/checkin

2. An entity of content type multipart/related, with two parts, an entity of content type application/checkin, and a second entity, of any content type, containing a check in body.

If the request-body is of content type multipart/related, the checkin operation is defined to be a PUT of the checkin entity body (into a server-dependent temporary resource), followed by a CHECKIN.

The definition of content type application/checkin is:

CheckIn = Schema-Version CRLF [Derived-From CRLF] [Working-Resource CRLF] [Request-Version CRLF]
Request-Version specifies a client’s request to set the version identifier of the check in resource. The server MAY set the version identifier of the check in resource to this value. If this field is omitted from the request message body, the server MUST set the version identifier if it has not previously been set.

Working-Resource specifies a URI or a version identifier of the working resource which is to be checked-in. If the value is "BODY," the working resource is contained in the check in body.

Open issues: 1 - If a version has multiple parents the client wants to know where the version links will go. If the system supports multiple inheritance then that needs to be stated. If the system only supports single inheritance then the client needs to know, before hand, which version the new version will be linked to. 2 - The syntax allows a client to specify different derived-from resources on Check In then were listed on Check Out. Somewhere there is a versioning system that doesn’t allow this. The client needs to know this fact before performing the check out.

7.4.4 Response Body

There is no response body defined for the CHECKIN method.

7.4.5 Error Conditions

If the Derived-From token(s) do not exist,

If the working URI is not located on this server, or the server is not authorized to retrieve the working URI.

If the PUT of the checkin entity body fails.

Syntax error in the request body.

7.5 DIFF/MERGE

7.5.1 Introduction

The request-URI of the DIFF method is an arbiter. The arbiter will compare the entries and return the differences between them.

The request-URI of the MERGE method is also an arbiter. The arbiter will combine the entries and return the result.
The Application/Diff content-type works with the DIFF method. The Application/Merge content-type works with the Merge method.

Application/Diff = Body
Application/Merge = Body
Body = 1#("URI" ":" URI | "Entity" ":" entity-body)

7.5.2 SERVERMERGE

7.5.2.1 Method Definition

The SERVERMERGE method requests that the server merge a number of entities on behalf of the client. The request-URI is a server side arbiter who performs the merge on behalf of the client.

7.5.2.2 Request Body

The request body may take on one of two possible formats. One format is application/servermerge, which is defined below. The second format is message/multipart related which will contain an entry of type application/servermerge as well as entities for the server to merge.

Type = application/servermerge

Merge = Schema-Version CRLF MergeType CRLF [MergeEntities CRLF] ReturnType CRLF [ReturnURI CRLF]
MergeType = "Merge Types" ":" ("URIs" | "Bodies" | "URIs" "Bodies")
MergeEntities = "Merge Entries" ":" #URI
ReturnType = "Return Result" ":" ("URI" | "Body")
ReturnURI = "Return Location" ":" URI

MergeType specifies what mechanisms are being used to refer to bodies to be merged. URIs means that a MergeEntities has been included while Bodies indicates that one or more message/multipart related have been included. ReturnType specifies how the results are to be returned, either as part of the response or recorded in a URI. ReturnURI specifies the URI for the result to be recorded in if the user selects that return mechanism. A server MAY support any combination of values for MergeType and ReturnType. It MUST provide for discovery of supported values.

7.5.2.3 Response Body

Only support # bodies to be merged

Can not resolve URI
Can not store in URI
Can not merge included types
Merge Failed
Merge Succeeded

7.5.2.4 Response Codes

[Will fold in from above.]

7.6 Command Comments

Command comments are entity headers.

Command_Comment = "Comment" ":" CommentVal

CommentVal = URI | comment

This is the standard comment facility used by versioning systems. Servers that do not understand the header or do not wish to make use of the information are free to ignore the information. No specification is made regarding how this information is to be retrieved. It is likely, however, that most systems will make the command comments available through their history attribute.

7.7 FREEZE (a.k.a. UNVERSION)

Editor: this method should be viewed as proposed functionality.

7.7.1 Method Definition

This method request the server to freeze the version tree in the current state, and copy the default published version outside of versioning space so that it is accessible as an unversioned resource to the client. After the version tree is frozen, the clients will not be able to perform any operation except CHECKIN at which time the tree will be taken out of the frozen state and put back into the version space. The Request-URI is the URI of the version tree handle.

7.7.2 Request Body

None.

INTERNET-DRAFT WEBDAV March 26, 1997

7.7.3 Response

The response will only consist of the status code as returned by the server.
7.7.4 Error Conditions

The error codes could any of the following:

OK - the server successfully froze the version tree and copied the Default Published Version into non-version space.
FREEZE failed because the server could not freeze the tree.
FREEZE failed because the server could not copy the default published version to unversioned space.
FREEZE failed because the server could not determine the Default Published Version.
The source (tree) is missing.

8. Ancillary Methods and Headers

8.1 MaxLength Header

8.1.1 Header Definition

The max length header requests that the server not return any results greater than specified in the header. The header also specifies if partial results are preferred to no results in the case when the result entity is greater than allowed by the header.

8.1.2 Header Body

MaxLengthHeader = "MaxLength" ":" Size Partial CRLF
Size = "Size" ":" Bytes-Unit 1*Digit
Partial = "Partial" ":" ("Acceptable" | "Not Acceptable")

8.2 If-Lock Header

8.2.1 Header Definition

The If-Lock header makes any request to which it has been appended conditional on the lock(s) referred to by the included token(s) still being active. This header can not be dropped.

8.2.2 Header Body

IfLockHeader = "If-Lock" ":" #LockToken CRLF

8.3 Command-Comments Header (CCMD)

8.3.1 Header Definition
Command-Comments (short form CCMD) is a request-header which is used to transmit comment information on any HTTP request method to the server. The server MAY provide persistent storage for the comments. The server MAY log the comments. The server MAY ignore the comments.

8.3.2 Header Body

The definition of the syntax of the Command-Comments header is as follows:

```
CommandComments = ("CCMD" | "Command-Comments") ":" LangTaggedString

LangTaggedString = Lang UTF-8 ; See RFC 2044 for UTF-8 definition
Lang = "(" LangTag ")"
LangTag = 1*8ALPHA *(""1*8ALPHA)
```

8.4 Overwrite Header

8.4.1 Header Definition

Overwrite is a request-header which specifies whether the destination of the method should or should not be overwritten.

8.4.2 Header Body

```
Overwrite = "Overwrite" ":" ("Yes" | "No")
```

If the value of Overwrite is "Yes," a destination resource will be replaced if it exists. If the value of Overwrite is "No," the server will not replace the destination resource if it exists.

This header may be applied to the PUT method, and thus modifies the definition of PUT given in Section 9.6 of [HTTP/1.1].

9. Required Features

[Ed. Note: preliminary and incomplete]

Introduce the concept of Distributed Authoring compliance, Versioning compliance, and totally optional features. Use the list from presentation.

10. Acknowledgements
11. References


[RFC1959]

[RFC1960]

[WEBC] - References to web collections.

Yaron Goland
Microsoft Corporation
One Microsoft Way
Bldg. 27N/3445
Redmond, WA 98052-6399
Fax (206) 936 7329
Email yarong@microsoft.com

E. J. Whitehead, Jr.
Dept. Of Information and Computer Science
University of California, Irvine
APPENDIX

A. Definition of LinkSearch Header
[Editor: This is a proposed feature.]
The LinkSearch method can be expensive, especially when a number of attributes must be retrieved over time.
A more economical mechanism is to append a search string to URL using "?". However, the URL could be confused with any of a number of search syntax’s. The solution proposed by this document is to create a new header called LinkSearch.
LinkSearch = "LinkSearch" "" HTTP_url ; See section 3.2.2 of [HTTP11]
If the HTTP URL does not end in a "/", then a "?" may be appended to it and the attribute search syntax defined below may be used to search for attributes on that particular resource. If the URL does end with a "/", then any resource whose name is a child of the specified HTTP URL may be searched on directly.
To perform a search a string matching the following syntax should be used. 

\[ \text{LinkSearchSuffix} = '?' \text{ Attribute} \]
\[ \text{Attribute} = \text{token} \]

So, if the LinkSearch value is \text{http://foo/bar} then a search on a link type of author would be performed using \text{http://foo/bar?author}. If the returned value is \text{http://foo/bar/} then the author search may be performed on \text{http://foo/bar/?author} and any of its children, such as \text{http://foo/bar/doublebar?author}.

By returning the LinkSearch header the server is agreeing to manipulate the search URL as it would the URL of the resource it resolves to. Thus it becomes the responsibility of the server to resolve the search URL whenever it is included in a command.

If it is impossible to resolve the search request to a single result then a 416 Not Unique error should be returned. This error specifies that the URI was not resolvable to a unique resource.

If the search URI does not resolve to any attribute then any request containing that URI should return a 404 Not Found error.

\textbf{B. Definition of LinkPrefix Header}

Editor: this is a proposed feature.

In some cases a server may have an arbiter which handles all searches on resources. To allow for the use of the arbiter but still gain the advantages of URL based searching the LinkPrefix is introduced.

\[ \text{LinkPrefix} = "\text{LinkPrefix}" "\text{:}" \text{HTTP_url} \]

Thus the returned URL has the URL to be searched on appended to it followed by the actual search syntax as previously specified. This allows for quick retrieval of attribute values but with the benefit of processing requests at a single resource.

The URI returned by LinkPrefix has the same use semantics as the URI returned by LinkSearch.