WEBDAV Working Group
Y.Y. Goland, Microsoft

INTERNET DRAFT
<draft-ietf-webdav-protocol-05>
E.J. Whitehead, Jr., UC Irvine
A. Faizi, Netscape
S.R. Carter, Novell
D. Jensen, Novell

Expires April, 1998
November 19, 1997

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

Status of this Memo

Abstract

This document specifies a set of methods, headers, and content-types ancillary to HTTP/1.1 for the management of resource properties, creation and management of resource collections, namespace manipulation, resource locking (collision avoidance), and efficient transmission of resource changes.

Changes

1.1. Changes since draft-ietf-webdav-protocol-04.txt

[Editor's note: This section will not appear in the final form of this document. Its purpose is to provide a concise list of changes

from the previous revision of the draft for use by reviewers.]

Added this change section.

Removed scoping for namespaces so the namespace for every element is explicitly stated.

Changed the syntax from <?XML:Namespace.../> to <?namespace...?>.

Removed propfindresult, this was left over from the old search format.

Changed all the DAV XML element names to lower case.

Changed the property format to use Name and Namespace rather than name and schema.

Removed proploc attribute and removed section on GETting, DELETEing, and PUTing properties since we do not provide a mechanism for getting a URI for properties. Also removed the requirement that properties be URI addressable.

Removed quoted string choice from owner header, it is just XML.

Made all the HTTP error codes use the same format.

Changed the name of the create element in PROPPATCH to set, the new name seems to cause less confusion.

Moved all headers in the draft to a single section.

Deleted the state token section of the draft and moved the state token headers to the header section of the draft. Removed the state token header.

Changed the write lock section to state that a Lock-Token request header, not a state-token request header, is to be submitted on request for write locked resources.

Created a "generic" XML element section for XML elements that get repeatedly re-used throughout the spec. I moved LINK XML element to this section.

Made multistatus and Schema discovery their own level one sections.

Collected all the properties together.

Removed all references to the possibility of properties have their own URIs. This includes removing the property identifier section.

Separated the section on web collections and namespaces into two separate sections.

Collected all the new response codes together into their own section.

Changed the XML multiresponse element name to multistatus.

Added a stand alone section on levels of DAV compliance. I also went method by method, property by property, to specify compliance requirements.

Added an introduction.

Changed all the "True" and "False" to "T" and "F".

Altered the first two paragraphs of the Property Names section to make the relationship between a property's name and its schema a little clearer. I also added some text in the same section defining a property name as a namespace and element.

Added a second paragraph to property model for http resources - overview. This paragraph clarifies why XML was chosen.

Added a 409 Conflict error to move to cover attempts to move a collection with members.

Changed the collection requirement to read the collections SHOULD end with "/". Also added a SHOULD about returning a location header if the client submits a URL for a collection without a trailing "/".

Moved the owner header into the body due to size concerns.

Replaced the iscollection xml element with resourcetype.

Moved the DAV property to the DAV header that is returned with OPTIONS.

Folded the tree draft into this draft. Changed the DELETE, COPY, and MOVE sections to include their effect on collections as taken from the tree draft. Created a Depth header section and put in the general rules that were in the introduction to the tree draft. I also added the 102 response and response-status header.

Removed the versioning section.

Put all the methods into a single section.

Replaced the PROPFIND request body with a propfind header. Now the response can be cached just using vary.

Nuked resinfo for INDEX and combined it with multistatus which is now used for both INDEX and PROPFIND. Stripped down INDEX as agreed.

Removed the problem definition and proposed solution sections. We can always cut and paste them together from the older version if we feel we need them but this draft is supposed to be a dry run for

last call and last call documents do not have problem definition/proposed solution sections.

Killed the section on schema discovery, it is controversial and we aren't going to be able to require it. We should specify it in a different spec.

Added a section on notational conventions used within the document.

Moved the terminology section to the end of the document to provide better flow from the high-level introduction to the specific introduction sections.

Increased the numeric value of the 4xx status codes introduced in this specification to avoid conflicts with the new revision of the HTTP/1.1 specification, which introduces two new 4xx status codes.

Wrote internationalization concerns section.

Added XML version number to all examples.

Contents

2. Introduction

This document describes an extension to the HTTP/1.1 protocol that allows clients to perform remote web content authoring operations. This extension provides a coherent set of methods, headers, request entity body formats, and response entity body formats that provide operations for:

Properties: The ability to create, remove, and query information about Web pages, such as its author, creation date, etc. Also, the ability to link pages of any media type to related pages.

Collections: The ability to create sets of related documents, and to receive a listing of pages at a particular hierarchy level (like a directory listing in a file system).

Locking: The ability to keep more than one person from working on a document at the same time. This prevents the "lost update problem" in which modifications are lost as first one author, then another writes their changes without merging the other author's changes

Namespace Operations: The ability to copy and move Web resources

Efficient Update: The ability to send changes which are proportional to the size of the change rather than retransmitting the entire resource.

Requirements and rationale for these operations are described in a companion document, "Requirements for a Distributed Authoring and Versioning Protocol for the World Wide Web" [Slein et al., 1997].

The sections below provide a detailed introduction to resource properties (Section 3), collections of resources (Section 4), and

locking operations (Section 5). These sections introduce the abstractions manipulated by the WebDAV-specific HTTP methods described in Section 8, "HTTP Methods for Distributed Authoring".

In HTTP/1.1, method parameter information was exclusively encoded in HTTP headers. Unlike HTTP/1.1, WebDAV, encodes method parameter information either in an Extensible Markup Language (XML) [Bray, Sperberg-McQueen, 1997] request entity body, or in an HTTP header. The use of XML to encode method parameters was motivated by the ability to add extra XML elements to existing structures, providing extensibility, and by XML's ability to encode information in ISO 10646 character sets, providing internationalization support. As a rule of thumb, parameters are encoded in XML entity bodies when they have unbounded length, or when they may be shown to a human user and hence require encoding in an ISO 10646 character set. Otherwise, parameters are encoded within an HTTP header. Section 9 describes the new HTTP headers used with WebDAV methods.

In addition to encoding method parameters, XML is used in WebDAV to encode the responses from methods, providing the extensibility and internationalization advantages of XML for method output, as well as input. XML elements used in this specification are defined in Section 12.

While the response codes provided by HTTP/1.1 are sufficient to describe the preponderance of error conditions encountered by WebDAV methods, there are some errors that do not fall neatly into the existing categories. New status codes developed for the WebDAV methods are defined in Section 10. Since some WebDAV methods may operate over many resources, the multiresponse status type has been introduced to return status information for multiple resources. Multiresponse status is described in Section 11.

The properties mechanism is employed by WebDAV to store information about the current state of the resource. For example, when a lock is taken out on a resource, a lock information property describes the current state of the lock. Section 13 defines the properties used within the WebDAV specification.

Finishing off the specification are sections on what it means to be compliant with this specification (Section 14), on internationalization support (Section 15), and on security (Section 16).

3. Data Model for Resource Properties

3.1. The Resource Property Model

Properties are pieces of data that describe the state of a resource. Properties are data about data.

Properties are used in distributed authoring environments to provide for efficient discovery and management of resources. For example, a 'subject' property might allow for the indexing of all resources by their subject, and an 'author' property might allow for the discovery of what authors have written which documents.

The DAV property model consists of name/value pairs. The name of a property identifies the property's syntax and semantics, and provides an address by which to refer to that syntax and semantics.

There are two categories of properties: "live" and "non-live". A live property has its syntax and semantics enforced by the server. This represents the two cases of a) the value of a property is read-only, maintained by the server, and b) the value of the property is maintained by the client, but server performs syntax checking on submitted values. A non-live property has its syntax and semantics enforced by the client; the server merely records the value of the property verbatim.

3.2. Existing Metadata Proposals

Properties have long played an essential role in the maintenance of large document repositories, and many current proposals contain some notion of a property, or discuss web metadata more generally. These include PICS [Miller et al., 1996], PICS-NG, the Rel/Rev draft [Maloney, 1996], Web Collections, XML [Bray, Sperberg-McQueen, 1997], several proposals on representing relationships within HTML, digital signature manifests (DCMF), and a position paper on Web metadata architecture [Berners-Lee, 1997]. Work on PICS-NG and Web Collections has been subsumed by the Resource Definition Framework (RDF) metadata activity of the World Wide Web Consortium, which consists of a network-based data model and an XML representation of that model.

Some proposals come from a digital library perspective. These include the Dublin Core [Weibel et al., 1995] metadata set and the Warwick Framework [Lagoze, 1996], a container architecture for different metadata schemas. The literature includes many examples of metadata, including MARC [MARC, 1994], a bibliographic metadata format, and RFC 1807 [Lasher, Cohen, 1995], a technical report bibliographic format employed by the Dienst system. Additionally, the proceedings from the first IEEE Metadata conference describe many community-specific metadata sets.

Participants of the 1996 Metadata II Workshop in Warwick, UK

[Lagoze, 1996], noted that, "new metadata sets will develop as the networked infrastructure matures" and "different communities will propose, design, and be responsible for different types of metadata." These observations can be corroborated by noting that many community-specific sets of metadata already exist, and there is significant motivation for the development of new forms of metadata as many communities increasingly make their data available in digital form, requiring a metadata format to assist data location and cataloging.

3.3. Properties and HTTP Headers

Properties already exist, in a limited sense, in HTTP message headers. However, in distributed authoring environments a relatively large number of properties are needed to describe the state of a resource, and setting/returning them all through HTTP headers is inefficient. Thus a mechanism is needed which allows a principal to identify a set of properties in which the principal is interested and to then set or retrieve just those properties.

3.4. Property Values

The value of a property is expressed as a well-formed XML document.

XML has been chosen because it is a flexible, self-describing, structured data format that supports rich schema definitions, and because of its support for multiple character sets. XML's self-describing nature allows any property's value to be extended by adding new elements. Older clients will not break because they will still have the data specified in the original schema and will ignore elements they do not understand. XML's support for multiple character sets allows human-readable properties to be encoded and read in a character set familiar to the user.

3.5. Property Names

A property name is a universally unique identifier that is associated with a schema that provides information about the syntax and semantics of the property.

Because a property's name is universally unique, clients can depend upon consistent behavior for a particular property across multiple resources, so long as that property is "live" on the resources in question.

The XML namespace mechanism, which is based on URIs, is used to name properties because it provides a mechanism to prevent namespace collisions and for varying degrees of administrative control.

The property namespace is flat; that is, no hierarchy of properties is explicitly recognized. Thus, if a property A and a property A/B exist on a resource, there is no recognition of any relationship between the two properties. It is expected that a separate specification will eventually be produced which will address issues relating to hierarchical properties.

Finally, it is not possible to define the same property twice on a single resource, as this would cause a collision in the resource's property namespace.

4. Collections of Web Resources

This section provides a description of a new type of Web resource, the collection, and discusses its interactions with the HTTP URL namespace. The purpose of a collection resource is to model collection-like objects (e.g., filesystem directories) within a server's namespace.

All DAV compliant resources MUST support the HTTP URL namespace model specified herein.

4.1. Collection Resources

A collection is a resource whose state consists of an unordered list of internal members, an unordered list of external members, and a

set of properties. An internal member resource MUST have a URI that is immediately relative to the base URI of the collection, that is, a relative URI in which "../" is illegal, which MUST begin with "./" and which SHOULD contain a "/" at the end of the URI if the internal member resource is itself a collection.

An external member resource MUST be an absolute URI that is not an internal URI. Any given internal or external URI MUST only belong to the collection once, i.e., it is illegal to have multiple instances of the same URI in a collection. Properties defined on collections behave exactly as do properties on non-collection resources.

There is a standing convention that when a collection is referred to by its name without a trailing slash, the trailing slash is automatically appended. Due to this, a resource MAY accept a URI without a trailing "/" to point to a collection. In this case it SHOULD return a location header in the response pointing to the URL ending with the "/". For example, if a client performs an INDEX on http://foo.bar/blah (no trailing slash), the resource http://foo.bar/blah/ (trailing slash) MAY respond as if the

operation were invoked on it, and SHOULD return a location header with http://foo.bar/blah/ in it.

4.2. Creation and Retrieval of Collection Resources

This document specifies the MKCOL method to create new collection resources, rather than using the existing HTTP/1.1 PUT or POST method, for the following reasons

In HTTP/1.1, the PUT method is defined to store the request body at the location specified by the Request-URI. While a description format for a collection can readily be constructed for use with PUT, the implications of sending such a description to the server are undesirable. For example, if a description of a collection that omitted some existing resources were PUT to a server, this might be interpreted as a command to remove those members. This would extend PUT to perform DELETE functionality, which is undesirable since it changes the semantics of PUT, and makes it difficult to control DELETE functionality with an access control scheme based on methods.

While the POST method is sufficiently open-ended that a _create a collection_ POST command could be constructed, this is undesirable because it would be difficult to separate access control for collection creation from other uses of POST.

This document specifies the INDEX method for listing the contents of a collection, rather than relying on the existing HTTP/1.1 GET method. This is to avoid conflict with the de-facto standard practice of redirecting a GET request on a directory to its index.html resource.

The exact definition of the behavior of GET and PUT on collections is defined later in this document.

4.3. HTTP URL Namespace Model

The HTTP URL Namespace is a hierarchical namespace where the hierarchy is delimited with the "/" character. DAV compliant resources MUST maintain the consistency of the HTTP URL namespace. Any attempt to create a resource (excepting the root member of a namespace) that would not be the internal member of a collection MUST fail. For example, if the collection http://www.foo.bar.org/a/ exists, but http://www.foo.bar.org/a/b/does not exist, an attempt to create http://www.foo.bar.org/a/b/c must fail.

4.4. Source Resources and Output Resources

For many resources, the entity returned by a GET method exactly matches the persistent state of the resource, for example, a GIF file stored on a disk. For this simple case, the URL at which a resource is accessed is identical to the URL at which the source (the persistent state) of the resource is accessed. This is also the case for HTML source files that are not processed by the server prior to transmission.

However, the server can sometimes process HTML resources before they are transmitted as a return entity body. For example, server-side-include directives within an HTML file instruct a server to replace the directive with another value, such as the current date. In this case, what is returned by GET (HTML plus date) differs from the persistent state of the resource (HTML plus directive). Typically there is no way to access the HTML resource containing the unprocessed directive.

Sometimes the entity returned by GET is the output of a data-producing process that is described by one or more source resources (that may not even have a location in the URL namespace). A single data-producing process may dynamically generate the state of a potentially large number of output resources. An example of this is a CGI script that describes a "finger" gateway process that maps part of the namespace of a server into finger requests, such as http://www.foo.bar.org/finger_gateway/user@host.

In the absence of distributed authoring capabilities, it is acceptable to have no mapping of source resource(s) to the URI namespace. In fact, preventing access to the source resource(s) has desirable security benefits. However, if remote editing of the source resource(s) is desired, the source resource(s) should be given a location in the URI namespace. This source location should not be one of the locations at which the generated output is retrievable, since in general it is impossible for the server to differentiate requests for source resources from requests for

process output resources. There is often a many-to-many relationship between source resources and output resources.

On WebDAV compliant servers, for all output resources which have a single source resource (and that source resource has a URI), the URI of the source resource SHOULD be stored in a link on the output resource with type http://www.ietf.org/standards/dav/source. Note that by storing the source URIs in links on the output resources, the burden of discovering the source is placed on the authoring client.

5. Locking

The ability to lock a resource provides a mechanism for serializing access to that resource. Using a lock, an authoring client can provide a reasonable guarantee that another principal will not modify a resource while it is being edited. In this way, a client can prevent the "lost update" problem.

This specification allows locks to vary over two client-specified parameters, the number of principals involved (exclusive vs. shared) and the type of access to be granted. Furthermore, this document only provides the definition of locking for one lock access type, the write lock. However, the syntax is extensible, and permits the eventual specification of other access types.

5.1. Exclusive Vs. Shared Locks

The most basic form of lock is an exclusive lock. This is a lock where the access right in question is only granted to a single principal. The need for this arbitration results from a desire to avoid having to constantly merge results.

However, there are times when the goal of a lock is not to exclude others from exercising an access right but rather to provide a mechanism for principals to indicate that they intend to exercise their access right. Shared locks are provided for this case. A shared lock allows multiple principals to receive a lock. Hence any principal with appropriate access can get the lock.

With shared locks there are two trust sets that affect a resource. The first trust set is created by access permissions. Principals who are trusted, for example, may have permission to write the resource. Those who are not, don't. Among those who have access permission to write the resource, the set of principals who have taken out a shared lock also must trust each other, creating a (typically) smaller trust set within the access permission write set.

Starting with every possible principal on the Internet, in most situations the vast majority of these principals will not have write

access to a given resource. Of the small number who do have write access, some principals may decide to guarantee their edits are free from overwrite conflicts by using exclusive write locks. Others may decide they trust their collaborators will not overwrite their work (the potential set of collaborators being the set of principals who have write permission) and use a shared lock, which informs their collaborators that a principal is potentially working on the resource.

The WebDAV extensions to HTTP do not need to provide all of the communications paths necessary for principals to coordinate their activities. When using shared locks, principals may use any out of band communication channel to coordinate their work (e.g., face-to-face interaction, written notes, post-it notes on the screen, telephone conversation, Email, etc.) The intent of a shared lock is to let collaborators know who else is potentially working on a resource.

Shared locks are included because experience from web distributed authoring systems has indicated that exclusive write locks are often too rigid. An exclusive write lock is used to enforce a particular editing process: take out exclusive write lock, read the resource, perform edits, write the resource, release the lock. This editing process has the problem that locks are not always properly released, for example when a program crashes, or when a lock owner leaves without unlocking a resource. While both timeouts and administrative action can be used to remove an offending lock, neither mechanism may be available when needed; the timeout may be long or the administrator may not be available.

Despite their potential problems, exclusive write locks are extremely useful, since often a guarantee of freedom from overwrite conflicts is what is needed. This specification provides both exclusive write locks and the less strict mechanism of shared locks.

5.2. Required Support

A WebDAV compliant server is not required to support locking in any form. If the server does support locking it MAY choose to support any combination of exclusive and shared locks for any access types.

The reason for this flexibility is that locking policy strikes to the very heart of the resource management and versioning systems employed by various storage repositories. These repositories require control over what sort of locking will be made available. For example, some repositories only support shared write locks while others only provide support for exclusive write locks while yet others use no locking at all. As each system is sufficiently different to merit exclusion of certain locking features, this specification leaves locking as the sole axis of negotiation within WebDAV.

5.3. Lock Tokens

A lock token is a URI that identifies a particular lock. A lock token is returned by every successful LOCK operation in the lock-token response header, and can also be discovered through lock

discovery on a resource.

Lock token URIs are required to be unique across all resources for all time. This uniqueness constraint allows lock tokens to be submitted across resources and servers without fear of confusion.

This specification provides a lock token URI scheme called opaquelocktoken that meets the uniqueness requirements. However resources are free to return any URI scheme so long as it meets the uniqueness requirements.

5.4. opaquelocktoken Lock Token URI Scheme

The opaquelocktoken URI scheme is designed to be unique across all resources for all time. Due to this uniqueness quality, a client MAY submit an opaque lock token in a Lock-Token request header and an if-state[-not]-match header on a resource other than the one that returned it.

All resources MUST recognize the opaquelocktoken scheme and, at minimum, recognize that the lock token was not generated by the resource. Note, however, that resources are not required to generate opaquelocktokens in LOCK method responses.

In order to guarantee uniqueness across all resources for all time the opaquelocktoken requires the use of the GUID mechanism.

Opaquelocktoken generators, however, have a choice of how they create these tokens. They can either generate a new GUID for every lock token they create, which is potentially very expensive, or they can create a single GUID and then add extension characters. If the second method is selected then the program generating the extensions MUST guarantee that the same extension will never be used twice with the associated GUID.

```
Opaque-Lock-Token = "opaquelocktoken" ":" GUID [Extension]
GUID = ; As defined in [Leach, Salz, 1997]
Extension = *urlc   ;urlc is defined in [Berners-Lee et al., 1997]
(draft-fielding-url-syntax-07.txt)
```

5.5. Lock Capability Discovery

Since server lock support is optional, a client trying to lock a resource on a server can either try the lock and hope for the best, or perform some form of discovery to determine what lock capabilities the server supports. This is known as lock capability discovery. Lock capability discovery differs from discovery of

supported access control types, since there may be access control

types without corresponding lock types. A client can determine what lock types the server supports by retrieving the supportedlock property.

Any DAV compliant resource that supports the LOCK method MUST support the supportedlock property.

5.6. Active Lock Discovery

If another principal locks a resource that a principal wishes to access, it is useful for the second principal to be able to find out who the first principal is. For this purpose the lockdiscovery property is provided. This property lists all outstanding locks, describes their type, and provides their lock token.

Any DAV compliant resource that supports the LOCK method MUST support the lockdiscovery property.

6. Write Lock

This section describes the semantics specific to the write access type for locks. The write lock is a specific instance of a lock type, and is the only lock type described in this specification. A DAV compliant resource MAY support the write lock.

6.1. Methods Restricted by Write Locks

A write lock prevents a principal without the lock from successfully executing a PUT, POST, PATCH, PROPPATCH, MOVE, DELETE, MKCOL, ADDREF or DELREF on the locked resource. All other current methods, GET in particular, function independent of the lock.

Note, however, that as new methods are created it will be necessary to specify how they interact with a write lock.

6.2. Write Locks and Properties

While those without a write lock may not alter a property on a resource it is still possible for the values of live properties to change, even while locked, due to the requirements of their schemas. Only dead properties and live properties defined to respect locks are guaranteed not to change while write locked.

6.3. Write Locks and Null Resources

It is possible to assert a write lock on a null resource in order to lock the name. Please note, however, that locking a null resource effectively makes the resource non-null, as the resource now has lock related properties defined on it.

6.4. Write Locks and Collections

A write lock on a collection prevents the addition or removal of members of the collection. As a consequence, when a principal issues a request to create a new internal member of a collection using PUT or POST, or to remove an existing internal member of a collection using DELETE, this request MUST fail if the principal does not have a write lock on the collection.

However, if a write lock request is issued to a collection containing internal member resources that are currently locked in a manner which conflicts with the write lock, the request MUST fail with a 409 Conflict status code.

6.5. Write Locks and COPY/MOVE

The owner of a write lock MUST NOT execute a MOVE method on a resource he has locked. This specification intentionally does not define what happens if a MOVE method request is made on a locked resource by the lock's owner.

A COPY method invocation MUST NOT duplicate any write locks active on the source.

6.6. Re-issuing Write Locks

If a principal already owns a write lock on a resource, any future requests for the same type of write lock, on the same resource, while the principal's previous write lock is in effect, MUST result in a successful response with the same lock token as provided for the currently existing lock. Two lock requests are defined to be identical if their Lock-Info headers are identical.

6.7. Write Locks and The Lock-Token Request Header

If a user agent is not required to have knowledge about a lock when requesting an operation on a locked resource, the following scenario might occur. Program A, run by User A, takes out a write lock on a resource. Program B, also run by User A, has no knowledge of the lock taken out by Program A, yet performs a PUT to the locked resource. In this scenario, the PUT succeeds because locks are associated with a principal, not a program, and thus program B, because it is acting with principal A's credential, is allowed to perform the PUT. However, had program B known about the lock, it would not have overwritten the resource, preferring instead to present a dialog box describing the conflict to the user. Due to this scenario, a mechanism is needed to prevent different programs from accidentally ignoring locks taken out by other programs with the same authorization.

In order to prevent these collisions the lock token request header is introduced. Please refer to the Lock Token Request Header section for details and requirements.

6.7.1. Write Lock Token Example

```
COPY /~fielding/index.html HTTP/1.1
Host: www.ics.uci.edu
Destination: http://www.ics.uci.edu/users/f/fielding/index.html
Lock-Token: <opaquelocktoken:123AbcEfg1284h23h2>
<opaquelocktoken:AAAASDFcalkjfdas12312>


HTTP/1.1 200 OK
```

In this example, both the source and destination are locked so two lock tokens must be submitted. If only one of the two resources was locked, then only one token would have to be submitted.

7. Notational Conventions

Since this document describes a set of extensions to the HTTP/1.1 protocol, the augmented BNF used herein to describe protocol elements is exactly the same as described in Section 2.1 of RFC 2068, _Hypertext Transfer Protocol -- HTTP/1.1_ [Fielding et al., 1997]. Since this augmented BNF uses the basic production rules provided in Section 2.2 of RFC 2068, these rules apply to this document as well.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [Bradner, 1997].

8. HTTP Methods for Distributed Authoring

8.1. PROPFIND

The PROPFIND method retrieves properties defined on the Request-URI, if it is a non-collection resource, or on the Request-URI and potentially its member resources, if the resource is a collection. All DAV compliant resources MUST support the PROPFIND method.

A client MAY submit a Depth header with a PROPFIND on a collection with a value of "0", "1" or "infinity". DAV compliant servers MUST support the "0", "1" and "infinity" behaviors. By default, the

PROPFIND method on a collection without a Depth header MUST act as if a Depth = infinity header was included.

A client MUST submit a Propfind request header describing what information is being requested. It is possible to request particular property values, all property values, or a list of the names of the resource's properties.

The response is a text/xml message body that contains a multistatus XML element that describes the results of the attempts to retrieve the various properties. If a property was successfully retrieved then its value MUST be returned in a prop XML element. If the scope of PROPFIND covers more than a single resource, as is the case with Depth values of "1" and "infinity", each response XML element MUST contain an href XML element which identifies the resource on which the properties in the prop XML element are defined. In the case of allprop and propname, if a principal does not have the right to know if a particular property exists, an error MUST NOT be returned. The results of this method SHOULD NOT be cached.

8.1.1. Example: Retrieving Named Properties

```
PROPFIND  /files/ HTTP/1.1
Host: www.foo.bar
Depth: 0
Propfind: <http://www.foo.bar/boxschema/bigbox> <http://www.foo.bar/
boxschema/author> <http://www.foo.bar/boxschema/DingALing> <http://w
ww.foo.bar/boxschema/Random>

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

<?XML version="1.0">
<?namespace href ="http://www.ietf.org/standards/dav/" AS = "D"?>
<?namespace href = "http://www.foo.bar/boxschema" AS = R"?>
<D:multistatus>
  <D:response>
       <D:prop>
            <R:bigbox>
                 <R:BoxType>Box type A</R:BoxType>
            </R:bigbox>
            <R:author>
                 <R:Name>J.J. Dingleheimerschmidt</R:Name>
            </R:author>
       </D:prop>
       <D:status>HTTP/1.1 200 OK</D:status>
  </D:response>
```

```
     <D:response>
          <D:prop><R:DingALing/><R:Random/></D:prop>
          <D:status>HTTP/1.1 403 Forbidden</D:status>
          <D:responsedescription> The user does not have access to the
   DingALing property.
          </D:responsedescription>
     </D:response>
```


```
     <D:responsedescription> There has been an access violation error.
     </D:responsedescription>
   </D:multistatus>
```

In this example, PROPFIND is executed on the collection http://www.foo.bar/files/. The specified depth is zero, hence the PROPFIND applies only to the collection itself, and not to any of its members. The Propfind header specifies the name of four properties whose values are being requested. In this case only two properties were returned, since the principal issuing the request did not have sufficient access rights to see the third and fourth properties.

8.1.2. Example: Using allprop to Retrieve All Properties

```
   PROPFIND  /container/ HTTP/1.1
   Host: www.foo.bar
   Depth: 1
   Propfind: allprop


   HTTP/1.1 200 OK
   Content-Type: text/xml
   Content-Length: xxxxx

   <?XML version="1.0">
   <?namespace href = "http://www.ietf.org/standards/dav/" As = "S"?>
   <?namespace href = "http://www.foo.bar/boxschema/" AS = R"?>
   <S:multistatus>
     <S:response>
          <S:href>http://www.foo.bar/container/</S:href>
          <S:prop>
               <R:bigbox>
                    <R:BoxType>Box type A</R:BoxType>
               </R:bigbox>
               <R:author>
                    <R:Name>Hadrian</R:Name>
               </R:author>
          </S:prop>
          <S:status>HTTP 1.1 200 OK</S:status>
```

```
     </S:response>
     <S:response>
          <S:href>http://www.foo.bar/container/index.html</S:href>
          <S:prop>
               <R:bigbox>
                    <R:BoxType>Box type B</R:BoxType>
               </R:bigbox>
          </S:prop>
          <S:status>HTTP 1.1 200 OK</S:status>
     </S:response>
   </S:multistatus>
```


In this example, PROPFIND was invoked on the resource http://www.foo.bar/container/ with a Depth header of 1, meaning the request applies to the resource and its children, and a Propfind header of "allprop", meaning the request should return the name and value of all properties defined on each resource.

The resource http://www.foo.bar/container/ has two properties defined on it, named http://www.foo.bar/boxschema/bigbox, and http://www.foo.bar/boxschema/author, while resource http://www.foo.bar/container/index.html has only a single resource defined on it, named http://www.foo.bar/boxschema/bigbox, another instance of the "bigbox" property type.

8.1.3. Example: Using propname to Retrieve all Property Names

```
PROPFIND  /container/ HTTP/1.1
Host: www.foo.bar
Propfind: propname

HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: xxxx

<?XML version="1.0">
<?namespace href = "http://www.ietf.org/standards/dav/" As = "D"?>
<?namespace href = "http://www.foo.bar/boxschema/" AS = "R"?>
<D:multistatus>
  <D:response>
       <D:href>http://www.foo.bar/container/</D:href>
       <D:prop>
            <R:bigbox/>
            <R:author/>
       </D:prop>
       <D:status>HTTP 1.1 200 OK</D:status>
  </D:response>
  <D:response>
```

```
        <D:href>http://www.foo.bar/container/index.html</D:href>
        <D:prop>
             <R:bigbox/>
        </D:prop>
        <D:status>HTTP 1.1 200 OK</D:status>
    </D:response>
</D:multistatus>
```

In this example, PROPFIND is invoked on the collection resource http://www.foo.bar/container/, with a Propfind header set to "propname", meaning the name of all properties should be returned. Since no depth header is present, it assumes its default value of "infinity", meaning the name of the properties on the collection and all its progeny should be returned.

Consistent with the previous example, resource http://www.foo.bar/container/ has two properties defined on it, http://www.foo.bar/boxschema/bigbox, and http://www.foo.bar/boxschema/author. The resource http://www.foo.bar/container/index.html, a member of the "container" collection, has only one property defined on it, http://www.foo.bar/boxschema/bigbox.

8.2. PROPPATCH

The PROPPATCH method processes instructions specified in the request body to set and/or remove properties defined on the resource identified by Request-URI.

All DAV compliant resources MUST support the PROPPATCH method and MUST process instructions that are specified using the propertyupdate, set, and remove XML elements of the DAV schema. Execution of the directives in this method is, of course, subject to access control constraints. DAV compliant resources MUST support the setting of arbitrary dead properties.

The request message body of a PROPPATCH method MUST contain at least one propertyupdate XML element. Instruction processing MUST occur in the order instructions are received (i.e., from top to bottom), and MUST be performed atomically.

8.2.1. propertyupdate XML element

Name: propertyupdate
Namespace: http://www.ietf.org/standards/dav/
Purpose: To contain a request to alter the properties on a resource.

Parent: None
Values= 1*(set | remove)
Description: This XML element is a container for the information required to modify the properties on the resource. This XML element is multi-valued.

8.2.2. set XML element

Name: set
Namespace: http://www.ietf.org/standards/dav/
Purpose: To set the DAV properties specified inside the set XML element.
Parent: propertyupdate
Values= prop
Description: This XML element MUST contain only a prop XML element. The elements contained by prop specify the name and value of properties that are set on the Request-URI. If a property already exists then its value is replaced.

8.2.3. remove XML element

Name: remove
Namespace: http://www.ietf.org/standards/dav/
Purpose: To remove the DAV properties specified inside the remove XML element.
Parent: propertyupdate
Values= prop
Description: Remove specifies that the properties specified in prop should be removed. Specifying the removal of a property that does not exist is not an error. All the elements in prop MUST be empty, as only the names of properties to be removed are required.

8.2.4. Response Codes

200 OK - The command succeeded. As there can be a mixture of sets and removes in a body, a 201 Create seems inappropriate.

403 Forbidden - The client, for reasons the server chooses not to specify, cannot alter one of the properties.

405 Conflict - The client has provided a value whose semantics are not appropriate for the property. This includes trying to set read-only properties.

413 Request Entity Too Long - If a particular property is too long to be recorded then a composite XML error will be returned indicating the offending property.

8.2.5. Example

```
   PROPPATCH /bar.html HTTP/1.1
   Host: www.foo.com
   Content-Type: text/xml
   Content-Length: xxxx

   <?XML version="1.0">
   <?namespace href = "http://www.ietf.org/standards/dav/" AS = "D"?>
   <?namespace href = "http://www.w3.com/standards/z39.50/" AS = "Z"?>
   <D:propertyupdate>
     <D:set>
          <D:prop>
               <Z:authors>
                    <Z:Author>Jim Whitehead</Z:Author>
                    <Z:Author>Roy Fielding</Z:Author>
               </Z:authors>
          </D:prop>
     </D:set>
     <D:remove>
          <D:prop><Z:Copyright-Owner/></D:prop>
     </D:remove>
   </D:propertyupdate>
```



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

   <?XML version="1.0">
   <?namespace href="http://www.ietf.org/standards/dav/" AS = "D"?>
   <?namespace href="http://www.w3.com/standards/z39.50/" AS = "Z"?>
   <D:multistatus>
     <D:response>
          <D:prop><Z:Authors/></D:prop>
          <D:status>HTTP/1.1 420 Method Failure</D:status>
     </D:response>
     <D:response>
          <D:prop><Z:Copyright-Owner/></D:prop>
          <D:status>HTTP/1.1 409 Conflict</D:status>
     </D:response>
     <D:responsedescription> Copyright Owner can not be deleted or
   altered.</D:responsedescription>
   </D:multistatus>
```

In this example, the client requests the server to set the value of the http://www.w3.com/standards/z39.50/Authors property, and to remove the property http://www.w3.com/standards/z39.50/Copyright-

Owner. Since the Copyright-Owner property could not be removed, no property modifications occur. The Method Failure response code for the Authors property indicates this action would have succeeded if it were not for the conflict with removing the Copyright-Owner property.

8.3. MKCOL Method

The MKCOL method is used to create a new collection. All DAV compliant resources MUST support the MKCOL method.

8.3.1. Request

MKCOL creates a new collection resource at the location specified by the Request-URI. If the Request-URI exists, then MKCOL must fail. During MKCOL processing, a server MUST make the Request-URI a member of its parent collection. If no such ancestor exists, the method MUST fail. When the MKCOL operation creates a new collection resource, all ancestors MUST already exist, or the method MUST fail with a 409 Conflict status code. For example, if a request to create collection /a/b/c/d/ is made, and neither /a/b/ nor /a/b/c/ exists, the request MUST fail.

When MKCOL is invoked without a request body, the newly created collection has no members.

A MKCOL request message MAY contain a message body. The behavior of a MKCOL request when the body is present is limited to creating collections, members of a collection, bodies of members and properties on the collections or members. If the server receives a MKCOL request entity type it does not support or understand it MUST respond with a 415 Unsupported Media Type status code. The exact behavior of MKCOL for various request media types is undefined in this document, and will be specified in separate documents.

8.3.2. Response Codes

Responses from a MKCOL request are not cacheable, since MKCOL has non-idempotent semantics.

201 Created - The collection or structured resource was created in its entirety.

403 Forbidden - This indicates at least one of two conditions: 1) The server does not allow the creation of collections at the given location in its namespace, and 2) The parent collection of the Request-URI exists but cannot accept members.

405 Method Not Allowed - MKCOL can only be executed on a

deleted/non-existent resource.

409 Conflict - A collection cannot be made at the Request-URI until one or more intermediate collections have been created.

415 Unsupported Media Type- The server does not support the request type of the body.

419 Insufficient Space on Resource - The resource does not have sufficient space to record the state of the resource after the execution of this method.

8.3.3. Example

This example creates a collection called /webdisc/xfiles/ on the server www.server.org.

```
MKCOL /webdisc/xfiles/ HTTP/1.1
Host: www.server.org


HTTP/1.1 201 Created
```

8.4. INDEX Method

The INDEX method is used to enumerate the members of a resource. All DAV compliant resources MUST support the INDEX method if they have members.

8.4.1. The Request

For a collection, INDEX MUST return a list of its members. All WebDAV compliant resources MUST support the text/xml response entity described below. The INDEX result for a collection MAY also return a list of the members of child collections, to any depth.

Collections that respond to an INDEX method with a text/xml entity MUST contain a single multistatus XML element which contains a response XML element for each member.

A resource that supports INDEX MUST return the resourcetype property for each member.

Note that the prop XML element MAY contain additional properties.

8.4.2. Example

```
INDEX /user/yarong/dav_drafts/ HTTP/1.1
Host: www.microsoft.com
```

```
HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: xxx
Last-Modified: Thu, 11 Sep 1997 23:45:12 GMT
ETag: _fooyyybar_

<?XML version="1.0">
<?namespace href = _http://www.ietf.org/standards/dav/_ as = _D_?>
<D:multistatus>
  <D:response>
       <D:href>http://www.microsoft.com/user/yarong/dav_drafts/
       </D:href>
       <D:prop>
            <D:resourcetype>
                 <D:collection/>
            </D:resourcetype>
       </D:prop>
       <D:status>HTTP 1.1 200 OK</D:status>
  </D:response>
  <D:response>
       <D:href>
       http://www.microsoft.com/user/yarong/dav_drafts/base
       </D:href>
       <D:prop>
            <D:resourcetype/>
       </D:prop>
       <D:status>HTTP 1.1 200 OK</D:status>
  </D:response>
</D:multistatus>
```


8.5. ADDREF Method

The ADDREF method is used to add external members to a resource. All DAV compliant collection resources MUST support the ADDREF method. All other DAV compliant resources MAY support the ADDREF method as appropriate.

8.5.1. The Request

The ADDREF method adds the URI specified in the Collection-Member header as an external member to the collection specified by the Request-URI. The value in the Collection-Member header MUST be an absolute URI meeting the requirements of an external member URI.

It is not an error if the URI specified in the Collection-Member header already exists as an external member of the collection. However, after processing the ADDREF there MUST be only one instance

of the URI in the collection. If the URI specified in the Collection-Member header already exists as an internal member of the collection, the ADDREF method MUST fail with a 412 Precondition Failed status code.

8.5.2. Example

```
ADDREF /~ejw/dav/ HTTP/1.1
Host: www.ics.uci.edu
Collection-Member: http://www.ietf.org/standards/dav/


HTTP/1.1 200 OK
```

This example adds the URI http://www.ietf.org/standards/dav/ as an external member resource of the collection http://www.ics.uci.edu/~ejw/dav/.

8.6. DELREF Method

The DELREF method is used to remove external members from a resource. All DAV compliant collection resources MUST support the DELREF method. All other DAV compliant resources MUST support the DELREF method only if they support the ADDREF method.

8.6.1. The Request

The DELREF method removes the URI specified in the Collection-Member header from the collection specified by the Request-URI.

DELREFing a URI which is not a member of the collection is not an error. DELREFing an internal member MUST fail with a 412 Precondition Failed status code.

8.6.2. Example

```
DELREF /~ejw/dav/ HTTP/1.1
Host: www.ics.udi.edu
Collection-Member: http://www.ietf.org/standards/dav/


HTTP/1.1 200 OK
```

This example removes the URI http://www.ietf.org/standards/dav/, an external member resource, from the collection http://www.ics.uci.edu/~ejw/dav/.

8.7. GET, HEAD for Collections

The semantics of GET are unchanged when applied to a collection, since GET is defined as, _retrieve whatever information (in the form of an entity) is identified by the Request-URI_ [Fielding et al., 1997]. GET when applied to a collection MAY return the contents of an _index.html_ resource, a human-readable view of the contents of the collection, or something else altogether, and hence it is possible the result of a GET on a collection will bear no correlation to the state of the collection.

Similarly, since the definition of HEAD is a GET without a response message body, the semantics of HEAD are unmodified when applied to collection resources.

8.8. POST for Collections

Since by definition the actual function performed by POST is determined by the server and often depends on the particular resource, the behavior of POST when applied to collections cannot be meaningfully modified because it is largely undefined. Thus the semantics of POST are unmodified when applied to a collection.

8.9. DELETE

8.9.1. DELETE Method for Non-Collection Resources

If the DELETE method is issued to a non-collection resource which is an internal member of a collection, then during DELETE processing a server MUST remove the Request-URI from its parent collection. A server MAY remove the URI of a deleted resource from any collections of which the resource is an external member.

8.9.2. DELETE for Collections

The DELETE method on a collection MUST act as if a Depth = Infinity header was used on it. A client MUST NOT submit a Depth header on a DELETE on a collection with any value but Infinity.

DELETE 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 cannot be deleted then all of the member's progeny MUST NOT be deleted, so as to maintain the namespace.

Any headers included with DELETE 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.

When the DELETE method has completed processing it MUST return a consistent namespace.

The response SHOULD be a Multi-Status response that describes the result of the DELETE on each affected resource.

8.9.2.1. Example

```
DELETE  /container/ HTTP/1.1
Host: www.foo.bar
Destroy: NoUndelete


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

<?XML version="1.0">
<?namespace href = "http://www.ietf.org/standards/dav/" As = "d"?>
<d:multistatus>
  <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 OK</d:status>
  </d:response>
  <d:response>
       <d:href>http://www.foo.bar/container/</d:href>
       <d:status>HTTP/1.1 420 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:multistatus>
```


In this example the attempt to delete http://www.foo.bar/container/resource3 failed because the server was unable to guarantee that resource3 would not be able to be undeleted. Consequently, the attempt to delete http://www.foo.bar/container/ also failed, but resource1 and resource2 were deleted. Even though a Depth header has not been included, a depth of infinity is assumed because the method is on a collection. As this example illustrates, DELETE processing need not be atomic.

8.10. PUT

8.10.1. PUT for Non-Collection Resources

A PUT performed on an existing resource replaces the GET response entity of the resource. Properties defined on the resource MAY be recomputed during PUT processing. For example, if a server recognizes the content type of the request body, it may be able to automatically extract information that could be profitably exposed as properties.

A PUT that would result in the creation of a resource without an appropriately scoped parent collection MUST fail with a 405 Method Not Allowed.

8.10.2. PUT for Collections

As defined in the HTTP/1.1 specification [Fielding et al., 1997], the "PUT method requests that the enclosed entity be stored under the supplied Request-URI." Since submission of an entity representing a collection would implicitly encode creation and deletion of resources, this specification intentionally does not define a transmission format for creating a collection using PUT. Instead, the MKCOL method is defined to create collections. If a PUT is invoked on a collection resource it MUST fail.

When the PUT operation creates a new non-collection resource all ancestors MUST already exist. If all ancestors do not exist, the method MUST fail with a 409 Conflict status code. For example, if resource /a/b/c/d.html is to be created and /a/b/c/ does not exist, then the request must fail.

8.11. COPY Method

The COPY method creates a duplicate of the specified resource. All DAV compliant resources MUST support the COPY method.

Support for the COPY method does not guarantee the ability to copy a resource. For example, separate programs may control resources on the same server. As a result, it may not even be possible to copy a resource to a location that appears to be on the same server.

8.11.1. The Request

The COPY method creates a duplicate of the source resource, given by the Request-URI, in the destination resource, given by the Destination header. The Destination header MUST be present. The exact behavior of the COPY method depends on the type of the source resource.

8.11.1.1. COPY for HTTP/1.1 resources

When the source resource is not a collection the body of the destination resource MUST be octet-for-octet identical to the body of the source resource. Alterations to the destination resource do not modify the source resource. Alterations to the source resource do not modify the destination resource. Thus, all copies are performed _by-value_.

All properties on the source resource MUST be duplicated on the destination resource, subject to modifying headers, following the definition for copying properties.

8.11.1.2. COPY for Properties

The following section defines how properties on a resource are handled during a COPY operation.

Live properties SHOULD be duplicated as identically behaving live properties at the destination resource. Since they are live properties, the server determines the syntax and semantics of these properties. Properties named by the Enforce-Live-Properties header MUST be live on the destination resource, or the method MUST fail. If a property is not named by Enforce-Live-Properties and cannot be copied live, then its value MUST be duplicated, octet-for-octet, in an identically named, dead property on the destination resource.

If a property on the source already exists on the destination resource and the Overwrite header is set to "T" then the property at the destination MUST be overwritten with the property from the source. If the Overwrite header is "F" and the previous situation exists, then the COPY MUST fail with a 409 Conflict.

8.11.1.3. COPY for Collections

The COPY method on a collection without a Depth header MUST act as if a Depth = infinity header was included. A client MAY submit a Depth header on a COPY on a collection with a value of "0" or "infinity". DAV compliant servers MUST support the "0" and "infinity" behaviors.

A COPY of depth infinity 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.

A COPY of depth "0" only instructs that the collection, the properties, and its external members, not its internal members, are

to be copied.

Any headers included with a COPY 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 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.

The response is a Multi-Status response that describes the result of the COPY on each affected 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 whether 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 421 Destination Locked error.

8.11.1.4. Type Interactions

If the destination resource identifies a collection and the Overwrite header is _T_, prior to performing the copy the server MUST perform a DELETE operation on the collection.

8.11.2. Response Codes

200 OK - The source resource was successfully copied to a pre-existing destination resource.

201 Created - The source resource was successfully copied. The copy operation resulted in the creation of a new resource.

412 Precondition Failed - This status code MUST be returned if the server was unable to maintain the liveness of the properties listed

in the Enforce-Live-Properties header, or if the Overwrite header is "F", and the state of the destination resource is non-null.

419 Insufficient Space on Resource - The destination resource does not have sufficient space to record the state of the resource after the execution of this method.

421 Destination Locked _ The destination resource was locked and either a valid Lock-Token header was not submitted, or the Lock-Token header identifies a lock held by another principal.

500 Server Error - The resource was in such a state that it could not be copied. This may occur if the Destination header specifies a resource that is outside the namespace the resource is able to interact with.

8.11.3. Overwrite Example

This example shows resource http://www.ics.uci.edu/~fielding/index.html being copied to the location http://www.ics.uci.edu/users/f/fielding/index.html. The contents of the destination resource were overwritten, if non-null.

```
COPY /~fielding/index.html HTTP/1.1
Host: www.ics.uci.edu
Destination: http://www.ics.uci.edu/users/f/fielding/index.html


HTTP/1.1 200 OK
```

8.11.4. No Overwrite Example

The following example shows the same copy operation being performed, except with the Overwrite header set to _F._ A response of 412 Precondition Failed is returned because the destination resource has a non-null state.

```
COPY /~fielding/index.html HTTP/1.1
Host: www.ics.uci.edu
Destination: http://www.ics.uci.edu/users/f/fielding/index.html
Overwrite: _F_


HTTP/1.1 412 Precondition Failed
```

8.11.5. Collection Example

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

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

<?XML version="1.0">
<?namespace href = "http://www.ietf.org/standards/dav/" As = "d"?>
<d:multistatus>
  <d:response>
       <d:href>http://www.foo.bar/othercontainer/resource1</d:href>
       <d:href>http://www.foo.bar/othercontainer/resource2</d:href>
       <d:href>http://www.foo.bar/othercontainer/</d:href>
       <d:href>http://www.foo.bar/othercontainer/R2/D2</d:href>
       <d:status>HTTP/1.1 201 Created</d:status>
  </d:response>
  <d:response>
       <d:href>http://www.foo.bar/othercontainer/R2/</d:href>
       <d:status>HTTP/1.1 412 Precondition Failed</d:status>
  </d:response>
</d:multistatus>
```

The Depth header is unnecessary as the default behavior of COPY on a collection is to act as if a "Depth: Infinity" header had been submitted. In this example most of the resources, along with the collection, were copied successfully. However the collection R2 failed, most likely due to a problem with enforcing live properties. R2's member D2 was successfully copied. As a result a collection was created at www.foo.bar/othercontainer/R2 to contain D2.

8.12. MOVE Method

The move operation on a resource is the logical equivalent of a copy followed by a delete, where the actions are performed atomically. All DAV compliant resources MUST support the MOVE method.

However, support for the MOVE method does not guarantee the ability to move a resource to a particular destination. For example, separate programs may actually control different sets of resources on the same server. Therefore, it may not even be possible to move a resource within a namespace that appears to belong to the same server.

8.12.1. The Request

If a resource exists at the destination, the destination resource will be DELETEd as a side effect of the MOVE operation, subject to the restrictions of the Overwrite header.

8.12.2. MOVE for Collections

MOVE 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.

The MOVE method on a collection MUST act as if a Depth "infinity" header was used on it. A client MUST NOT submit a Depth header on a MOVE on a collection with any value but "infinity".

Any headers included with MOVE 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 on collections.

When the MOVE 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.

The response is a Multi-Status response that describes the result of the MOVE 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 whether 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 421 Destination Locked error.

8.12.3. Response Codes

200 OK - The move operation was successful.

409 Conflict _ The MOVE was attempted on a collection with members. While the COPY part of this operation could succeed the DELETE could not. Therefore the MOVE MUST fail.

412 Precondition Failed - This status code MUST be returned if the server was unable to maintain the liveness of the properties listed in the Enforce-Live-Properties header, or if the Overwrite header is "F", and the state of the destination resource is non-null.

421 Destination Locked - The destination resource was locked and either a valid Lock-Token header was not submitted, or the Lock-Token header identifies a lock held by another principal.

502 Bad Gateway - This may occur when the destination is

o

n another server and the destination server refuses to accept the resource

8.12.4. Overwrite Example

This example shows resource http://www.ics.uci.edu/~fielding/index.html being moved to the location http://www.ics.uci.edu/users/f/fielding/index.html. The contents of the destination resource were overwritten, if non-null.

```
MOVE /~fielding/index.html HTTP/1.1
Host: www.ics.uci.edu
Destination: http://www.ics.uci.edu/users/f/fielding/index.html


HTTP/1.1 200 OK
```

8.12.5. Collection Example

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


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

<?XML version="1.0">
<?namespace href = "http://www.ietf.org/standards/dav/" As = "D"?>
<d:multistatus>
  <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/1.1 420 Method Failure</d:status>
    <d:response>
         <d:href>http://www.foo.bar/othercontainer/C2</d:href>
         <d:status>HTTP/1.1 409 Conflict</d:status>
    </d:response>
  </d:multistatus>
```

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 moved, although its child container/C2/R2 could be moved.

8.13. LOCK Method

The following sections describe the LOCK method, which is used to take out a lock of any access type. These sections on the LOCK method describe only those semantics that are specific to the LOCK method and are independent of the access type of the lock being requested. Once the general LOCK method has been described, subsequent sections describe the semantics of the "write" access type, and the write lock.

8.13.1. Operation

A LOCK method invocation creates the lock specified by the Lock-Info header on the Request-URI. Lock method requests SHOULD have a XML request body which contains an Owner XML element for this lock request. The LOCK request MAY have a Timeout header.

A successful response to a lock invocation MUST include Lock-Token and Timeout headers. Clients MUST assume that locks may arbitrarily disappear at any time, regardless of the value given in the Timeout header. The Timeout header only indicates the behavior of the server if "extraordinary" circumstances do not occur. For example, an administrator may remove a lock at any time or the system may crash in such a way that it loses the record of the lock's existence. The response MUST also contain the value of the lockdiscovery property in a prop XML element.

8.13.2. The Effect of Locks on Properties and Collections

By default the scope of a lock is the entire state of the resource,

including its body and associated properties. As a result, a lock on a resource also locks the resource's properties, and a lock on a property may lock a property's resource or may restrict the ability to lock the property's resource. Only a single lock token MUST be used when a lock extends to cover both a resource and its properties. Note that certain lock types MAY override this behavior.

For collections, a lock also affects the ability to add or remove members. The nature of the effect depends upon the type of access control involved.

8.13.3. Locking Replicated Resources

Some servers automatically replicate resources across multiple URLs. In such a circumstance the server MAY only accept a lock on one of

the URLs if the server can guarantee that the lock will be honored across all the URLs.

8.13.4. Locking Multiple Resources

The LOCK method supports locking multiple resources simultaneously by allowing for the listing of several URIs in the LOCK request. These URIs, in addition to the Request-URI, are then to be locked as a result of the LOCK method's invocation. When multiple resources are specified the LOCK method only succeeds if all specified resources are successfully locked.

The Lock-Tree option of the lock request specifies that the resource and all its internal children (including internal collections, and their internal members) are to be locked. This is another mechanism by which a request for a lock on multiple resources can be specified.

Currently existing locks can not be extended to cover more or less resources, and any request to expand or contract the number of resources in a lock MUST fail with a 409 Conflict status code. So, for example, if resource A is exclusively write locked and then the same principal asks to exclusively write lock resources A, B, and C, the request will fail as A is already locked and the lock can not be extended.

A successful result will return a single lock token which represents all the resources that have been locked. If an UNLOCK is executed on this token, all associated resources are unlocked.

If the lock cannot be granted to all resources, a 409 Conflict status code MUST be returned with a response entity body containing

a multistatus XML element describing which resource(s) prevented the lock from being granted.

8.13.5. Interaction with other Methods

The interaction of a LOCK with various methods is dependent upon the lock type. However, independent of lock type, a successful DELETE of a resource MUST cause all of its locks to be removed.

8.13.6. Lock Compatibility Table

The table below describes the behavior that occurs when a lock request is made on a resource.

Current lock state/ Lock request	Shared Lock	Exclusive Lock
None	True	True
Shared Lock	True	False
Exclusive Lock	False	False*

Legend: True = lock MAY be granted. False = lock MUST NOT be granted. *=if the principal requesting the lock is the owner of the lock, the lock MAY be regranted.

The current lock state of a resource is given in the leftmost column, and lock requests are listed in the first row. The intersection of a row and column gives the result of a lock request. For example, if a shared lock is held on a resource, and an exclusive lock is requested, the table entry is _false_, indicating the lock must not be granted.

If an exclusive or shared lock is re-requested by the principal who owns the lock, the lock MUST be regranted. If the lock is regranted, the same lock token that was previously issued MUST be returned.

8.13.7. Owner XML Element

Name: owner
Namespace: http://www.ietf.org/standards/dav/
Purpose: Provide information about the principal taking out a lock.
Parent: Any
Values: XML Elements
Descripton: The Owner XML element provides information sufficient for either directly contacting a principal (such as a telephone number or Email URI), or for discovering the principal (such as the

URL of a homepage) who owns a lock.

8.13.8. Lock Response

A successful lock response MUST contain a Lock-Token response header, a Timeout header and a prop XML element in the response body which contains the value of the lockdiscovery property.

8.13.9. Response Codes

409 Conflict - The resource is locked, so the method has been rejected.

412 Precondition Failed - The included Lock-Token was not enforceable on this resource or the server could not satisfy the request in the Lock-Info header.

8.13.10. Example - Simple Lock Request

```
LOCK /workspace/webdav/proposal.doc HTTP/1.1
Host: webdav.sb.aol.com
Lock-Info: LockType=Write LockScope=Exclusive
Timeout: Infinite; Second-4100000000
Content-Type: text/xml
```



```
Content-Length: xyz

<?XML version="1.0">
<?namespace href="http://www.ietf.org/standards/dav/" AS = "D"?>
<D:owner>
   <D:href>http://www.ics.uci.edu/~ejw/contact.html</D:href>
</D:owner>


HTTP/1.1 200 OK
Lock-Token: opaquelocktoken:xyz122393481230912asdfa09s8df09s7df
Timeout: Second-604800
Content-Type: text/xml
Content-Length: xxxxx

<?XML version="1.0">
<?namespace href ="http://www.ietf.org/standards/dav/" AS = "D"?>
<D:prop>
  <D:lockdiscovery>
       <D:activelock>
            <D:locktype>write</D:locktype>
            <D:lockscope>exclusive</D:lockscope>
            <D:addlocks/>
            <D:owner>
```

```
                    <D:href>
                    http://www.ics.uci.edu/~ejw/contact.html
                    </D:href>
               </D:owner>
               <D:timeout>Second-604800</D:timeout>
               <D:locktoken>
                    <D:href>
               opaquelocktoken:xyz122393481230912asdfa09s8df09s7df
                    </D:href>
               </D:locktoken>
          </D:activelock>
     </D:lockdiscovery>
   </D:prop>
```

This example shows the successful creation of an exclusive write lock on resource http://webdav.sb.aol.com/workspace/webdav/proposal.doc. The resource http://www.ics.uci.edu/~ejw/contact.html contains contact information for the owner of the lock. The server has an activity-

based timeout policy in place on this resource, which causes the lock to automatically be removed after 1 week (604800 seconds). The response has a Lock-Token header that gives the lock token URL that uniquely identifies the lock created by this lock request.

8.13.11. Example - Multi-Resource Lock Request

```
   LOCK /workspace/webdav/proposal.doc HTTP/1.1
   Host: webdav.sb.aol.com
```



```
   Lock-Info: LockType=Write LockScope=Exclusive
   Addlocks=<http://webdav.sb.aol.com/workspace/><http://foo.bar/blah>
   Timeout: Infinite, Second-4100000000

   <?XML version="1.0">
   <?namespace href="http://www.ietf.org/standards/dav/" AS = "D"?>
   <D:href>http://www.ics.uci.edu/~ejw/contact.html<D:href>


   HTTP/1.1 409 Conflict
   Content-Type: text/xml
   Content-Length: xxxxx

   <?XML version="1.0">
   <?namespace href = "http://www.ietf.org/standards/dav/" As = "D"?>
   <D:multistatus>
     <D:response>
          <D:href>
               http://webdav.sb.aol.com/workspace/webdav/proposal.doc
```

```
        </D:href>
        <D:href>
             http://webdav.sb.aol.com/workspace/webdav/
        </D:href>
        <D:status>HTTP/1.1 202 Accepted</D:status>
    </D:response>
    <D:response>
        <D:href>http://foo.bar/blah</D:href>
        <D:status>HTTP/1.1 403 Forbidden</D:status>
    </D:response>
</D:multistatus>
```

This example shows a request for an exclusive write lock on three resources, http://webdav.sb.aol.com/workspace/webdav/proposal.doc, http://webdav.sb.aol.com/workspace/, and http://foo.bar/blah. In this request, the client has specified that it desires an infinite length lock, if available, otherwise a timeout of 4.1 billion seconds, if available. The Owner header field specifies the web address for contact information for the principal taking out the lock.

This lock request has failed, because the server rejected the lock request for http://foo.bar/blah. The 409 Conflict status code indicates that the server was unable to satisfy the request because there is a conflict between the state of the resources and the operation named in the request. Within the multistatus, the 202 Accepted status code indicates that the lock method was accepted by the resources, and would have been completed if all resources named in the request were able to be locked. The 403 Forbidden status code indicates that the server does not allow lock requests on this resource.

8.14. UNLOCK Method

The UNLOCK method removes the lock identified by the lock token in the Lock-Token header from the Request-URI, and all other resources included in the lock.

Any DAV compliant resource which supports the LOCK method MUST support the UNLOCK method.

8.14.1. Example

```
UNLOCK /workspace/webdav/info.doc HTTP/1.1
Host: webdav.sb.aol.com
Lock-Token:opaquelocktoken:123AbcEfg1284h23h2

HTTP/1.1 200 OK
```

In this example, the lock identified by the lock token "opaquelocktoken:123AbcEfg1284h23h2" is successfully removed from the resource http://webdav.sb.aol.com/workspace/webdav/info.doc. If this lock included more than just one resource, the lock was removed from those resources as well.

8.15. PATCH Method

The PATCH method is used to modify parts of the entity returned in the response to a GET method. DAV compliant resources MAY support the PATCH method.

8.15.1. The Request

The request entity of the PATCH method contains a list of differences between the resource identified by the Request-URI and the desired content of the resource after the PATCH action has been applied. The list of differences is in a format defined by the media type of the entity (e.g., "application/diff") and must include sufficient information to allow the server to convert the original version of the resource to the desired version. Processing performed by PATCH is atomic. Hence all changes MUST be successfully executed or the method fails. PATCH MUST fail if executed on a non-existent resource; i.e., PATCH does not create a resource as a side effect.

If the request appears (at least initially) to be acceptable, the server MUST transmit an interim 100 response message after receiving the empty line terminating the request headers and continue processing the request. Since the semantics of PATCH are non-idempotent, responses to this method are not cacheable.

While server support for PATCH is optional, if a server does support PATCH, it MUST support at least the text/xml diff format defined

below. Support for the VTML difference format [VTML] is recommended, but not required.

8.15.2. text/xml elements for PATCH

The resourceupdate XML element contains a set of XML sub-entities that describe modification operations. The name and meaning of these XML elements are given below. Processing of these directives MUST be performed in the order encountered within the XML document. A directive operates on the resource as modified by all previous directives (executed in sequential order). The length of the resource MAY be extended or reduced by a PATCH.

The changes specified by the resourceupdate XML element MUST be executed atomically.

8.15.2.1. resourceupdate XML Element

Name: resourceupdate
Namespace: http://www.ietf.org/standards/dav/patch/
Purpose: Contains an ordered set of changes to a non-collection, non-property resource.
Parent: None
Value= *(insert | delete | replace)

8.15.2.2. insert XML Element

Name: insert
Namespace: http://www.ietf.org/standards/dav/patch/
Purpose: Insert the XML element's contents starting at the specified octet.
Parent: resourceupdate
Value: The insert XML element MUST contain an octet-range XML attribute that specifies an octet position within the body of a resource. A value of _end_ specifies the end of the resource. The body of the insert XML element contains the octets to be inserted.

Please note that in order to protect the white space contained in this XML element the following attribute/value MUST be included in the element: XML-SPACE = "PRESERVE". This attribute is defined in the XML specification [Bray, Sperberg-McQueen, 1997].

8.15.2.3. delete XML Element

Name: delete
Namespace: http://www.ietf.org/standards/dav/patch/
Purpose: Removes the specified range of octets.
Parent: resourceupdate
Value: The delete XML element MUST contain an octet-range XML attribute.

Discussion: The octets that are deleted are removed, which means the resource is collapsed and the length of the resource is decremented by the size of the octet range. It is not appropriate to replace deleted octets with zeroed-out octets, since zero is a valid octet value.

8.15.2.4. replace XML Element

Name: replace
Namespace: http://www.ietf.org/standards/dav/patch/

Purpose: Replaces the specified range of octets with the contents of the XML element. If the number of octets in the XML element is different from the number of octets specified, the update MUST be rejected.
Parent: resourceupdate
Value: The replace XML element MUST contain an octet-range XML attribute. The contents of the entity are the replacement octets.

Please note that in order to protect the white space contained in this XML element the following attribute/value MUST be included in the element: XML-SPACE = "PRESERVE"

.

This attribute is defined in the XML specification [Bray, Sperberg-McQueen, 1997].

8.15.2.5. octet-range Attribute

Name: octet-range
Namespace: http://www.ietf.org/standards/dav/patch/
Purpose: Specifies a range of octets that the enclosing property affects.
Parent: insert | delete | replace
Value: number [_-_ (number | _end_)]
Number = 1*Digit

Description: Octet numbering begins with 0. If the octet contains a single number then the operation is to begin at that octet and to continue for a length specified by the operation. In the case of a delete, this would mean to delete a single octet. In the case of an insert this would mean to begin the insertion at the specified octet and to continue for the length of the included value, extending the resource if necessary. In the case of replace, the replace begins at the specified octet and overwrites all that follow to the length of the included value.

8.15.3. Response Codes

200 OK - The request entity body was processed without error, resulting in an update to the state of the resource.

409 Conflict - If the update information in the request message body does not make sense given the current state of the resource (e.g., an instruction to delete a non-existent line), this status code MAY be returned.

415 Unsupported Media Type - The server does not support the content type of the update instructions in the request message body.

418 Unprocessable Entity - The entity body submitted with the PATCH was not understood by the resource.

419 Insufficient Space on Resource - The resource does not have sufficient space to record the state of the resource after the execution of this method.

8.15.4. HTML file modification Example

The following example shows a modification of the title and contents of the HTML resource http://www.example.org/hello.html.

```
Before:
<HTML>
<HEAD>
<TITLE>Hello world HTML page</TITLE>
</HEAD>
<BODY>
<P>Hello, world!</P>
</BODY>
</HTML>
```

PATCH Request:

```
PATCH hello.html HTTP/1.1
Host: www.example.org
Content-Type: text/xml
Content-Length: xxx
```

Response:

```
HTTP/1.1 100 Continue
```

```
<?XML version="1.0">
<?namespace href = _http://www.ietf.org/standards/dav/patch/_ AS =
_D_?>
<D:resourceupdate>
  <D:replace XML-SPACE = "PRESERVE">
       <D:octet-range>14</D:octet-range>&003CTITLE&003ENew
Title&003C/TITLE&003E</D:replace>
  <D:delete><D:octet-range>38-50</D:octet-range></D:delete>
  <D:insert XML-SPACE = "PRESERVE"><D:octet-range>86</D:octet-
range>&003CP&003ENew paragraph&003C/P&003E</D:insert>
</D:resourceupdate>
```

```
HTTP/1.1 200 OK
```

```
After:
<HTML>
```

```
<HEAD>
<TITLE>New Title</TITLE>
</HEAD>
<BODY>
<P>Hello, world!</P>
<P>New paragraph</P>
</BODY>
</HTML>
```

9. HTTP Headers for Distributed Authoring

9.1. Collection-Member Header

```
CollectionMember = "Collection-Member" ":" URI   ; URI is defined in
section 3.2.1 of [Fielding et al., 1997]
```

The Collection-Member header specifies the URI of an external resource to be added/deleted to/from a collection.

9.2. DAV Header

```
DAV = "DAV" ":" ("1" | "2" | extend)
```

This header indicates that the resource supports the DAV schema and protocol to the level indicated. All DAV compliant resources MUST return the DAV header on all OPTIONS responses.

9.3. Depth Header

```
Depth = "Depth" ":" ("0" | "1" | "infinity")
```

The Depth header is used with methods executed on collections to indicate whether the method is to be applied only to the collection (Depth = 0), to the collection and its immediate children, (Depth = 1), or the collection and all its progeny (Depth = infinity). Note that Depth = 1 and Depth = infinity behavior only applies to internal member resources, and not to external member resources.

The Depth header is only supported if a method's definition explicitly provides for such support.

The following rules are the default behavior for any method that supports the depth header. A method MAY override these defaults by defining different behavior in its definition.

Methods which support the depth header MAY choose not to support all of the header's values and MAY define, on a case by case basis, the behavior of the method on a collection if a depth header is not present. For example, the MOVE method only supports Depth = infinity and if a depth header is not present will act as if a Depth =

Clients MUST NOT rely upon methods executing on members of their hierarchies in any particular order or the execution being atomic. Note that methods MAY provide guarantees on ordering and atomicity.

Upon execution, a method with a depth header will perform as much of its assigned task as possible and then return a response specifying what it was able to accomplish and what it failed to do.

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

Any headers on a method with a depth header 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.

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

9.4. Destination Header

Destination = "Destination" ":" URI

The Destination header specifies a destination resource for methods such as COPY and MOVE, which take two URIs as parameters.

9.5. Destroy Header

DestroyHeader = "Destroy" ":" #Choices

Choices = "VersionDestroy" | "NoUndelete" | "Undelete" | extend

Extend = RFC-Reg | Coded-URL

RFC-Req = Token ; This is a token value (defined in section 2.2 of [Fielding et al., 1997]) that has been published as an RFC.

Coded-URL = "<" URI ">"

When deleting a resource the client often wishes to specify exactly what sort of delete should be performed. The Destroy header, used with the Mandatory header, allows the client to specify the end result it desires. The Destroy header is specified as follows:

The Undelete token requests that, if possible, the resource should be left in a state such that it can be undeleted. The server is not required to honor this request.

The NoUndelete token requests that the resource MUST NOT be left in a state such that it can be undeleted.

The VersionDestroy token includes the functionality of the NoUndelete token and extends it to include having the server remove all versioning references to the resource that it has control over.

9.6. Enforce-Live-Properties Header

EnforceLiveProperties = "Enforce-Live-Properties_ _:" (_*_ | "Omit" | 1*(Property-Name))

Property-Name = Coded-URL

The Enforce-Live-Properties header specifies properties that MUST be _live_ after they are copied (moved) to the destination resource of a copy (or move). If the value _*_ is given for the header, then it designates all live properties on the source resource. If the value is "Omit" then the server MUST NOT duplicate on the destination resource any properties that are defined on the source resource. If this header is not included then the server is expected to act as

defined by the default property handling behavior of the associated method.

9.7. If-None-State-Match

If-None-State-Match = "If-None-State-Match" ":" 1#Coded-URL

The If-None-State-Match header is intended to have similar functionality to the If-None-Match header defined in section 14.26 of RFC 2068. However the if-none-state-match header is intended for use with any URI which represents state information about a resource, referred to as a state token. A typical example is a lock token.

If any of the state tokens identifies the current state of the resource, the server MUST NOT perform the requested method. Instead, if the request method was GET, HEAD, INDEX, or PROPFIND, the server SHOULD respond with a 304 Not Modified response, including the cache-related entity-header fields (particularly ETag) of the current state of the resource. For all other request methods, the server MUST respond with a status of 412 Precondition Failed.

If none of the state tokens identifies the current state of the resource, the server MAY perform the requested method.

If any of the tokens is not recognized then the method MUST fail with a 412 Precondition Failed.

Note that the "AND" and "OR" keywords specified with the If-State-Match header are intentionally not defined for If-None-State-Match, because this functionality is not required.

9.8. If-State-Match

```
If-State-Match = "If-State-Match" ":" ("AND" | "OR") 1#Coded-URL
```

The If-State-Match header is intended to have similar functionality to the If-Match header defined in section 14.25 of RFC 2068. However the If-State-Match header is intended for use with any URI which represents state information about a resource. A typical example is a lock token.

If the AND keyword is used and all of the state tokens identify the state of the resource, then the server MAY perform the requested method. If the OR keyword is used and any of the state tokens identifies the current state of the resource, then the server MAY perform the requested method. If the keyword requirement for the the keyword used is not met, the server MUST NOT perform the requested method, and MUST return a 412 Precondition Failed response.

If any of the tokens is not recognized then the method MUST fail with a 412 Precondition Failed.

9.9. Lock-Info Request Header

```
LockInfo = "Lock-Info" ":" DAVLockType SP DAVLockScope [SP
AdditionalLocks] [SP Lock-Tree]
DAVLockType = "LockType" "=" DAVLockTypeValue
DAVLockTypeValue = ("Write" | Extend)
DAVLockScope = "LockScope" "=" DAVLockScopeValue
DAVLockScopeValue = ("Exclusive" |"Shared" | Extend)
AdditionalLocks = "AddLocks" "=" 1*("<" URI ">")
Lock-Tree = "Lock-Tree" "=" ("T" | "F")
```

The Lock-Info request header specifies the scope and type of a lock for a LOCK method request. The syntax specification below is extensible, allowing new type and scope identifiers to be added.

The LockType field specifies the access type of the lock. At

present, this specification only defines one lock type, the "Write" lock. The LockScope field specifies whether the lock is an exclusive lock, or a shared lock. The AddLocks field specifies additional URIs, beyond the Request-URI, to which the lock request applies. The LockTree field is used to specify recursive locks. If the LockTree field is "T", the lock request applies to the hierarchy traversal of the internal member resources of the Request-URI, and the AddLocks URIs, inclusive of the Request-URI and the AddLocks URIs. It is not an error if LockTree is "T", and the Request-URI or the AddLocks URIs have no internal member resources. By default,

the value of LockTree is "F", and this field MAY be omitted when its value is "F".

9.10. Lock-Token Request Header

Lock-Token = "Lock-Token" ":" 1#Coded-URL

The Lock-Token request header, containing a lock token owned by the requesting principal, is used by the principal to indicate that the principal is aware of the existence of the lock specified by the lock token.

If the following conditions are met:

1) The method is restricted by a lock type that requires the submission of a lock token, such as a write lock,
2) The user-agent has authenticated itself as a principal,
3) The user-agent is submitting a method request to a resource on which the principal owns a write lock,

Then:

1) The method request MUST include a Lock-Token header with the lock token, or,
2) The method MUST fail with a 409 Conflict status code.

If multiple resources are involved with a method, such as a COPY or MOVE method, then the lock tokens, if any, for all involved resources, MUST be included in the Lock-Token request header.

For example, Program A, used by user A, takes out a write lock on a resource. Program A then makes a number of PUT requests on the locked resource. All the requests contain a Lock-Token request header that includes the write lock state token. Program B, also run by User A, then proceeds to perform a PUT to the locked resource. However, program B was not aware of the existence of the lock and so does not include the appropriate Lock-Token request header. The method is rejected even though principal A is

authorized to perform the PUT. Program B can, if it so chooses, now perform lock discovery and obtain the lock token. Note that programs A and B can perform GETs without using the Lock-Token header because the ability to perform a GET is not affected by a write lock.

Having a lock token provides no special access rights. Anyone can find out anyone else's lock token by performing lock discovery. Locks are to be enforced based upon whatever authentication mechanism is used by the server, not based on the secrecy of the token values.

9.11. Lock-Token Response Header

Lock-Token = "Lock-Token" ":" Coded-URL

If a resource is successfully locked then a Lock-Token header will be returned containing the lock token that represents the lock.

9.12. Overwrite Header

Overwrite = "Overwrite" ":" ("T" | "F")

The Overwrite header specifies whether the server should overwrite the state of a non-null destination resource during a COPY or MOVE. A value of _F_ states that the server MUST NOT perform the COPY or MOVE operation if the state of the destination resource is non-null. By default, the value of Overwrite is _T,_ and a client MAY omit this header from a request when its value is _T._ While the Overwrite header appears to duplicate the functionality of the If-Match: * header of HTTP/1.1, If-Match applies only to the Request-URI, and not to the Destination of a COPY or MOVE.

If a COPY or MOVE is not performed due to the value of the Overwrite header, the method MUST fail with a 409 Conflict status code.

9.13. Propfind Request Header

Propfind = "Propfind" ":" ("allprop" | "propname" | RFC-Reg | 1*(Property-Name))

The Propfind header is used to specify which properties are to be returned in a PROPFIND method. The properties are identified by their URIs. Two special tokens are defined for use with the Propfind header, allprop and propname. The allprop token specifies that all property names and values on the resource are to be returned. The propname token specifies that only a list of property names on the resource are to be returned.

9.14. 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 [Fielding et al., 1997]

The URIs listed in the header are source resources which have been affected by the outstanding method. The status code indicates the resolution of the method on the identified resource. So, for example, if a MOVE method on a collection is outstanding and a 102 "Processing" response with a Status-URI response header is returned, the included URIs will indicate resources that have had move attempted on them and what the result was.

9.15. Timeout Header

TimeOut = "Timeout" ":" 1#TimeType
TimeType = ("Second-" DAVTimeOutVal | "Infinite" | Other)
DAVTimeOutVal = 1*digit
Other = Extend field-value ; See section 4.2 of RFC 2068

Clients MAY include Timeout headers in their LOCK requests. However, the server is not required to honor or even consider these requests. Clients MUST NOT submit a Timeout request header with any method other than a LOCK method.

A Timeout request header MUST contain at least one TimeType and MAY contain multiple TimeType entries. The purpose of listing multiple TimeType entries is to indicate multiple different values and value types that are acceptable to the client. The client lists the TimeType entries in order of preference.

The Timeout response header MUST use a Second value, Infinite, or a TimeType the client has indicated familiarity with. The server MAY assume a client is familiar with any TimeType submitted in a Timeout header.

The _Second_ TimeType specifies the number of seconds that MUST elapse between granting of the lock at the server, and the automatic removal of the lock. A server MUST not generate a timeout value for _Second_ greater than 2^32-1.

The timeout counter is restarted any time an owner of the lock sends a method to any member of the lock, including unsupported methods, or methods which are unsuccessful. It is recommended that the HEAD method be used when the goal is simply to restart the timeout

counter.

If the timeout expires then the lock is lost. Specifically the server SHOULD act as if an UNLOCK method was executed by the server on the resource using the lock token of the timed-out lock, performed with its override authority. Thus logs should be updated with the disposition of the lock, notifications should be sent, etc., just as they would be for an UNLOCK request.

Servers are advised to pay close attention to the values submitted by clients, as they will be indicative of the type of activity the client intends to perform. For example, an applet running in a browser may need to lock a resource, but because of the instability of the environment within which the applet is running, the applet may be turned off without warning. As a result, the applet is likely to ask for a relatively small timeout value so that if the applet dies, the lock can be quickly harvested. However, a document management system is likely to ask for an extremely long timeout because its user may be planning on going off-line.

10. Response Code Extensions to HTTP/1.1

The following response codes are added to those defined in HTTP/1.1 [Fielding et al., 1997].

10.1. 102 Processing

Methods can potentially take a long period of time to process, especially methods that support the Depth header. 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 20 seconds (a reasonable, but arbitrary value) to process the server SHOULD return a 102 "Processing" response.

10.2. 207 Multi-Status

The response requires providing status for multiple independent operations.

10.3. 418 Unprocessable Entity

The server understands the content type of the request entity, but was unable to process the contained instructions.

10.4. 419 Insufficient Space on Resource

The resource does not have sufficient space to record the state of the resource after the execution of this method.

10.5. 420 Method Failure

The method was not executed on a particular resource within its scope because some part of the method's execution failed causing the entire method to be aborted. For example, if a resource could not be moved as part of a MOVE method, all the other resources would fail with a 420 Method Failure.

10.6. 421 Destination Locked

The destination resource of a method is locked, and either the request did not contain a valid Lock-Info header, or the Lock-Info header identifies a lock held by another principal.

11. Multi-Status Response

The default 207 Multi-Status response body is a text/xml HTTP entity that contains a single XML element called multistatus, which contains a set of XML elements called response, one for each 200,

300, 400, and 500 series status code generated during the method invocation. 100 series status codes MUST NOT be recorded in a response XML element.

11.1. multistatus XML Element

Name: multistatus
Namespace: http://www.ietf.org/standards/dav/
Purpose: Contains multiple response messages.
Parent: Any
Value: 1*response [responsedescription]
Description: The responsedescription at the top level is used to provide a general message describing the overarching nature of the response. If this value is available an application MAY use it instead of presenting the individual response descriptions contained within the responses.

11.2. response XML Element

Name: response
Namespace: http://www.ietf.org/standards/dav/
Purpose: Holds a single response
Parent: multistatus
Value: href [prop] status [responsedescription]
Description: Prop MUST contain one or more empty XML elements

representing the names of properties. Multiple properties may be included if the same response applies to them all. If href is used then the response refers to a problem with the referenced resource, not a property.

11.3. status XML Element

Name: status
Namespace: http://www.ietf.org/standards/dav/
Purpose: Holds a single HTTP status-line
Parent: response
Value: status-line ;status-line defined in [Fielding et al., 1997]

11.4. responsedescription XML Element

Name: responsedescription
Namespace: http://www.ietf.org/standards/dav/
Purpose: Contains a message that can be displayed to the user explaining the nature of the response.
Parent: multistatus | response
Value: Any
Description: This XML element provides information suitable to be presented to a user.

12. Generic DAV XML Elements

12.1. href XML Element

Name: href
Namespace: http://www.ietf.org/standards/dav/
Purpose: To identify that the content of the element is a URI.
Parent: Any
Value: URI ; See section 3.2.1 of [Fielding et al., 1997]

12.2. link XML Element

Name: link
Namespace: http://www.ietf.org/standards/dav/
Purpose: To identify a property as a link and to contain the source and destination of that link.
Values= 1*src 1*dst
Description: Link is used to provide the sources and destinations of a link. The type of the property containing the link XML element provides the type of the link. Link is a multi-valued element, so multiple Links may be used together to indicate multiple links with the same type.

12.2.1. src XML Element

Name: src
Namespace: http://www.ietf.org/standards/dav/
Purpose: To indicate the source of a link.
Parent: link
Values= URI

12.2.2. dst XML Element

Name: dst
Namespace: http://www.ietf.org/standards/dav/
Purpose: To indicate the destination of a link
Parent: link
Values= URI

12.2.3. Example

```
<?XML version="1.0">
<?namespace href = "http://www.ietf.org/standards/dav/" AS = "D"?>
<?namespace href = "http://www.foocorp.com/Project/" AS = "F"?>
<D:prop>
  <D:Source>
       <D:link>
            <F:projfiles>Source</F:projfiles>
            <D:src>http://foo.bar/program</D:src>
            <D:dst>http://foo.bar/src/main.c</D:dst>
       </D:link>
       <D:link>
            <F:projfiles>Library</F:projfiles>
```



```
            <D:src>http://foo.bar/program</D:src>
            <D:dst>http://foo.bar/src/main.lib</D:dst>
       </D:link>
       <D:link>
            <F:projfiles>Makefile</F:projfiles>
            <D:src>http://foo.bar/program</D:src>
            <D:dst>http://foo.bar/src/makefile</D:dst>
       </D:link>
  </D:Source>
</D:prop>
```

In this example the resource http://foo.bar/program has a source property that contains three links. Each link contains three elements, two of which, src and dst, are part of the DAV schema defined in this document, and one which is defined by the schema http://www.foocorp.com/project/ (Source, Library, and Makefile). A client which only implements the elements in the DAV spec will not

understand the foocorp elements and will ignore them, thus seeing the expected source and destination links. An enhanced client may know about the foocorp elements and be able to present the user with additional information about the links. This example demonstrates the power of XML markup that allows for element values to be enhanced without breaking older clients.

12.3. prop XML element

Name: prop
Namespace: http://www.ietf.org/standards/dav/
Purpose: Contains properties related to a resource.
Parent: Any
Values: XML Elements
Description: The prop XML element is a generic container for properties defined on resources. All elements inside prop MUST define properties related to the resource. No other elements may be used inside of a prop element.

13. DAV Properties

13.1. creationdate Property

Name: creationdate
Namespace: http://www.ietf.org/standards/dav/
Purpose: The time and date the resource was created.
Value: The time and date MUST be given in ISO 8601 format [ISO8601]
Description: This property SHOULD be defined on all DAV compliant resources. If present, it contains a timestamp of the moment when the resource was created (i.e., the moment it had non-null state).

13.2. displayname Property

Name: displayname
Namespace: http://www.ietf.org/standards/dav/
Purpose: A name for the resource that is suitable for presentation to a user.
Value: Any valid XML character data (as defined in [Bray, Sperberg-McQueen, 1997])
Description:This property SHOULD be defined on all DAV compliant resources. If present, the property contains a description of the resource that is suitable for presentation to a user.

13.3. get-content-language Property

Name: get-content-language
Namespace: http://www.ietf.org/standards/dav/

Purpose: Contains the Content-Language header returned by a GET without accept headers. If no Content-Language header is available, this property MUST NOT exist.
Value: language-tag ;language-tag is defined in section 14.13 of RFC 2068

13.4. get-content-length Property

Name: get-content-length
Namespace: http://www.ietf.org/standards/dav/
Purpose: Contains the Content-Length header returned by a GET without accept headers. If no Content-Length header is available, this property MUST NOT exist.
Value: content-length ; see section 14.14 of RFC 2068

13.5. get-content-type Property

Name: get-content-type
Namespace: http://www.ietf.org/standards/dav/
Purpose: Contains the Content-Type header returned by a GET without accept headers. If no Content-Type header is available, this property MUST NOT exist.
Value: media-type ; defined in Section 3.7 of [Fielding et al., 1997]

13.6. get-etag Property

Name: get-etag
Namespace: http://www.ietf.org/standards/dav/
Purpose: Contains the ETag header returned by a GET without accept headers. If no ETag header is available, this property MUST NOT exist.
Value: entity-tag ; defined in Section 3.11 of [Fielding et al., 1997]
Description:Note that the ETag on some resource may reflect changes in any part of the state of the resource, not necessarily just a change to the response to the GET method. For example, a change in the ACL may cause the ETag to change.

13.7. get-last-modified Property

Name: get-last-modified
Namespace: http://www.ietf.org/standards/dav/
Purpose: Contains the Last-Modified header returned by a GET method without accept headers. If no Last-Modified header is available, this property MUST NOT exist.
Value: HTTP-date ; defined in Section 3.3.1 of [Fielding et al., 1997]

Description:Note that the last-modified date on some resource may reflect changes in any part of the state of the resource, not necessarily just a change to the response to the GET method. For example, a change in a property may cause the last-modified date to change.

13.8. index-content-language Property

Name: index-content-language
Namespace: http://www.ietf.org/standards/dav/
Purpose: Contains the Content-Language header returned by an INDEX without accept headers. If no Content-Language header is available, this property MUST NOT exist.
Value: language-tag ;language-tag is defined in section 14.13 of RFC 2068

13.9. index-content-length Property

Name: index-content-length
Namespace: http://www.ietf.org/standards/dav/
Purpose: Contains the Content-Length header returned by an INDEX without accept headers. If no Content-Length header is available, this property MUST NOT exist.
Value: content-length ; see section 14.14 of RFC 2068

13.10. index-content-type Property

Name: index-content-type
Namespace: http://www.ietf.org/standards/dav/
Purpose: Contains the Content-Type header returned by an INDEX without accept headers. If no Content-Type header is available, this property MUST NOT exist.
Value: media-type ; defined in Section 3.7 of [Fielding et al., 1997]

13.11. index-etag Property

Name: index-etag
Namespace: http://www.ietf.org/standards/dav/
Purpose: Contains the ETag header returned by an INDEX without accept headers. If no ETag header is available, this property MUST NOT exist.

Value: entity-tag ; defined in Section 3.11 of [Fielding et al., 1997]
Description:Note that the ETag on some resource may reflect changes in any part of the state of the resource, not necessarily just a change to the response to the INDEX method. For example, a change in the ACL may cause the ETag to change.

13.12. index-last-modified Property

Name: index-last-modified
Namespace: http://www.ietf.org/standards/dav/
Purpose: Contains the Last-Modified header returned by an INDEX method without accept headers. If no Last-Modified header is available, this property MUST NOT exist.
Value: HTTP-date ; defined in Section 3.3.1 of [Fielding et al., 1997]
Description:Note that the last-modified date on some resource may reflect changes in any part of the state of the resource, not necessarily just a change to the response to the INDEX method. For example, a change in a property may cause the last-modified date to change.

13.13. lockdiscovery Property

Name: lockdiscovery
Namespace: http://www.ietf.org/standards/dav/
Purpose: To discover what locks are active on a resource
Values= *activelock
Description:The lockdiscovery property returns a listing of who has a lock, what type of lock he have, the timeout type and the time remaining on the timeout, and the associated lock token. The server is free to withhold any or all of this information if the requesting principal does not have sufficient access rights to see the requested data. A server which supports locks MUST provide the lockdiscovery property on any resource with locks on it.

13.13.1. activelock XML Element

Name: activelock
Namespace: http://www.ietf.org/standards/dav/
Purpose: A multivalued XML element that describes a particular active lock on a resource
Parent: lockdiscovery
Values= locktype lockscope [addlocks] owner timeout locktoken

13.13.2. owner XML Element

Name: owner
Namespace: http://www.ietf.org/standards/dav/
Purpose: Returns owner information
Parent: activelock
Values= XML:REF | *PCDATA

13.13.3. timeout XML Element

Name: timeout
Namespace: http://www.ietf.org/standards/dav/
Purpose: Returns information about the timeout associated with the lock
Parent: activelock
Values= TimeType

13.13.4. addlocks XML Element

Name: addlocks
Namespace: http://www.ietf.org/standards/dav/
Purpose: Lists additional resources associated with this lock, if any.
Parent: activelock
Values= 1*href

13.13.5. locktoken XML Element

Name: locktoken
Namespace: http://www.ietf.org/standards/dav/
Purpose: Returns the lock token
Parent: activelock
Values= href
Description:The href contains a Lock-Token-URL.

13.13.6. Example

```
PROPFIND /container/ HTTP/1.1
Host: www.foo.bar
Content-Length: xxxx
Content-Type: text/xml

<?XML version="1.0">
<?namespace href = "http://www.ietf.org/standards/dav/" AS = "D"?>
<D:propfind>
  <D:prop><lockdiscovery/></D:prop>
</D:propfind>

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

<?XML version="1.0">
<?namespace href ="http://www.ietf.org/standards/dav/" AS = "D"?>
<D:multistatus>
  <D:response>
       <D:prop>
            <D:lockdiscovery>
                 <D:activelock>
```

```
                        <D:locktype>write</D:locktype>
                        <D:lockscope>exclusive</D:lockscope>
                        <D:addlocks>
                             <D:href>http://foo.com/doc/</D:href>
                        </D:addlocks>
                        <D:owner>Jane Smith</D:owner>
                        <D:timeout>Infinite</D:timeout>
                        <D:locktoken>
                             <D:href>iamuri:unique!!!!!</D:href>
                        </D:locktoken>
                   </D:activelock>
              </D:lockdiscovery>
         </D:prop>
         <D:status>HTTP/1.1 200 OK</D:status>
    </D:response>
</D:multistatus>
```

This resource has a single exclusive write lock on it, with an infinite timeout. This same lock also covers the resource http://foo.com/doc/.

13.14. resourcetype Property

Name: resourcetype
Namespace: http://www.ietf.org/standards/dav/
Purpose: This property contains a series of XML elements that specify information regarding the nature of the resource. This specification only defines a single value, collection.
Value: XML elements
Description: This property MUST be defined on all DAV compliant resources. The default value is empty.

13.14.1. collection XML Element

Name: collection
Namespace: http://www.ietf.org/standards/dav/
Purpose: Identifies the associated resource as a collection. Collection resources MUST define this value with the resourcetype property.
Parent: resourcetype
Values: None

13.15. Source Link Property Type

Name: source
Namespace: http://www.ietf.org/standards/dav/link/
Purpose: The destination of the source link identifies the resource that contains the unprocessed source of the link's source.
Parent: None

Value: An XML document with zero or more link XML elements.

Discussion: The source of the link (src) is typically the URI of the output resource on which the link is defined, and there is typically only one destination (dst) of the link, which is the URI where the unprocessed source of the resource may be accessed. When more than

one link destination exists, this specification asserts no policy on ordering.

13.16. supportedlock Property

Name: supportedlock
Namespace: http://www.ietf.org/standards/dav/
Purpose: To provide a listing of the lock capabilities supported by the resource.
Values: An XML document containing zero or more LockEntry XML elements.
Description:The supportedlock property of a resource returns a listing of the combinations of scope and access types which may be specified in a lock request on the resource. Note that the actual contents are themselves controlled by access controls so a server is not required to provide information the client is not authorized to see. If supportedlock is available on _*_ then it MUST define the set of locks allowed on all resources on that server.

13.16.1. lockentry XML Element

Name: lockentry
Namespace: http://www.ietf.org/standards/dav/
Purpose: Defines a DAVLockType/LockScope pair that may be legally used with a LOCK on the specified resource.
Parent: supportedlock
Values= locktype lockscope

13.16.2. locktype XML Element

Name: locktype
Namespace: http://www.ietf.org/standards/dav/
Purpose: Lists a DAVLockType
Parent: lockentry
Values= DAVLockTypeValue

13.16.3. lockscope XML Element

Name: lockscope
Namespace: http://www.ietf.org/standards/dav/
Purpose: Lists a DAVLockScope
Parent: lockentry

Values: DAVLockScopeValue

13.16.4. Example

```
PROPFIND  /container/ HTTP/1.1
Host: www.foo.bar
```



```
Content-Length: xxxx
Content-Type: text/xml

<?XML version="1.0">
<?namespace href = "http://www.ietf.org/standards/dav/" AS = "D"?>
<D:propfind>
  <D:prop><supportedlock/></D:prop>
</D:propfind>

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

<?XML version="1.0">
<?namespace href ="http://www.ietf.org/standards/dav/" AS = "D"?>
<D:multistatus>
  <D:response>
       <D:prop>
            <D:supportedlock>
                 <D:LockEntry>
                      <D:locktype>Write</D:locktype>
                      <D:lockscope>Exclusive</D:lockscope>
                 </D:LockEntry>
                 <D:LockEntry>
                      <D:locktype>Write</D:locktype>
                      <D:lockscope>Shared</D:lockscope>
                 </D:LockEntry>
            </D:supportedlock>
       </D:prop>
       <D:status>HTTP/1.1 200 OK</D:status>
  </D:response>
</D:multistatus>
```

14. DAV Compliance Levels

A DAV compliant resource can choose from two levels of compliance. A client can discover which level a resource supports by executing OPTIONS on the resource, and examining the "DAV" header which is returned.

Since this document describes extensions to the HTTP/1.1 protocol, minimally all DAV compliant resources, clients, and proxies MUST be compliant with RFC 2068 [Fielding et al., 1997].

14.1. Level 1

A level 1 compliant resource MUST meet all "MUST" requirements in all sections of this document.

14.2. Level 2

A level 2 compliant resource MUST meet all level 1 requirements and support the supportedlock property as well as the LOCK method.

15. Internationalization Considerations

In the realm of internationalization issues, this specification is substantively in compliance with the IETF Character Set Policy [Alvestrand, 1997]. In this specification, human-readable fields can be found in either the value of a property, or in an error message returned in a response entity body. In both cases, the human-readable content is encoded using XML, which has explicit provisions for character set tagging and encoding, and requires by default that XML processors read XML elements encoded using the UTF-8 and UCS-2 encodings of the ISO 10646 basic multilingual plane. Furthermore, XML contains provisions for encoding XML elements using other encoding schemes, notable among them UCS-4, which permits encoding of characters from any ISO 10646 character plane.

The default character set encoding for XML data in this specification, and in general, is UTF-8. WebDAV compliant applications MUST support the UTF-8 and UCS-2 character set encodings for XML elements, and SHOULD support the UCS-4 encoding. The XML character set encoding declaration for each supported character set MUST also be supported, since it is by using this encoding declaration that an XML processor determines the encoding of an element.

XML also provides language tagging capability which provides the ability to specify the language of the contents of a particular XML element. Although XML, and hence WebDAV, does not use RFC 1766 language tags for its language names, the benefit of using standard XML in this context outweighs the advantage of using RFC 1766 language tags.

Names used within this specification fall into two categories: names specific to protocol elements such as methods and headers, names of XML elements, and names of properties. Naming of protocol elements

follows the precedent of HTTP, using English names encoded in USASCII for methods and headers. Since these protocol elements are not visible to users, and are in fact simply long token identifiers, they do not need to support encoding in multiple character sets. Similarly, though the names of XML elements used in this specification are English names encoded in UTF-8, these names are not visible to the user, and hence do not need to support multiple character set encodings.

The name of a property defined on a resource is a URI. Although some applications (e.g., a generic property viewer) will display property URIs directly to their users, it is expected that the typical application will use a fixed set of properties, and will provide a mapping from the property name URI to a human-readable

field when displaying the property name to a user. It is only in the case where the set of properties is not known ahead of time that an application need display a property name URI to a user. We recommend that applications provide human-readable property names wherever feasible.

For error reporting, we follow the convention of HTTP/1.1 status codes, including with each status code a short, English description of the code (e.g., 421 Destination Locked). While the possibility exists that a poorly crafted user agent would display this message to a user, internationalized applications will ignore this message, and display an appropriate message in the user's language and character set.

Since interoperation of clients and servers does not require locale information, this specification does not specify any mechanism for transmission of this information.

16. Security Considerations

[TBD]

17. Terminology

Collection - A resource that contains member resources.

Member Resource - A resource contained by a collection. There are two types of member resources: external and internal.

Internal Member Resource _ A member resource of a collection whose URI is relative to the URI of the collection.

External Member Resource - A member resource of a collection with an

absolute URI that is not relative to its parent's URI.

Property - A name/value pair that contains descriptive information about a resource.

Live Property _ A property whose semantics and syntax are enforced by the server. For example, a live "content-length" property would have its value, the length of the entity returned by a GET request, automatically calculated by the server.

Dead Property _ A property whose semantics and syntax are not enforced by the server. The server only records the value of a dead property; the client is responsible for maintaining the consistency of the syntax and semantics of a dead property.

18. Copyright

The following copyright notice is copied from RFC 2026 chapter 10.4, and describes the applicable copyright for this document

19. Acknowledgements

A specification such as this thrives on piercing critical review and withers from apathetic neglect. The authors gratefully acknowledge the contributions of the following people, whose insights were so valuable at every stage of our work.

Terry Allen, Harald Alvestrand, Alan Babich, Dylan Barrell, Bernard Chester, Tim Berners-Lee, Dan Connolly, Jim Cunningham, Ron Daniel, Jr., Jim Davis, Keith Dawson, Mark Day, Martin Duerst, David Durand, Lee Farrell, Chuck Fay, Roy Fielding, Mark Fisher, Alan Freier, George Florentine, Jim Gettys, Phill Hallam-Baker, Dennis Hamilton, Steve Henning, Alex Hopmann, Andre van der Hoek, Ben Laurie, Paul Leach, Ora Lassila, Karen MacArthur, Steven Martin, Larry Masinter, Michael Mealling, Keith Moore, Henrik Nielsen, Kenji Ota, Bob Parker, Glenn Peterson, Jon Radoff, Saveen Reddy, Henry Sanders, Christopher Seiwald, Judith Slein, Mike Spreitzer, Einar Stefferud, Ralph Swick, Kenji Takahashi, Richard N. Taylor, Robert Thau, John Turner, Sankar Virdhagriswaran, Fabio Vitali, Gregory Woodhouse, and Lauren Wood.

One from this list deserves special mention. The contributions by Larry Masinter have been invaluable, both in helping the formation of the working group and in patiently coaching the authors along the way. In so many ways he has set high standards we have toiled to meet.

20. References

[Alvestrand, 1997] H. T. Alvestrand, "IETF Policy on Character Sets and Languages." Internet-draft, work-in-progress. ftp://ds.internic.net/internet-drafts/draft-alvestrand-charset-policy-02.txt

[Berners-Lee, 1997] T. Berners-Lee, "Metadata Architecture." Unpublished white paper, January 1997. http://www.w3.org/pub/WWW/DesignIssues/Metadata.html.

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

[Bray, Sperberg-McQueen, 1997] T. Bray, C. M. Sperberg-McQueen, "Extensible Markup Language (XML): Part I. Syntax", WD-xml-lang.html, http://www.w3.org/pub/WWW/TR/WD-xml-lang.html.

[Fielding et al., 1997] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1." RFC 2068. U.C. Irvine, DEC, MIT/LCS. January, 1997. ftp://ds.internic.net/rfc/rfc2068.txt

[Lasher, Cohen, 1995] R. Lasher, D. Cohen, "A Format for Bibliographic Records," RFC 1807. Stanford, Myricom. June, 1995. ftp://ds.internic.net/rfc/rfc1807.txt

[Leach, Salz, 1997] P. J. Leach, R. Salz, "UUIDs and GUIDs." Internet-draft (expired), work-in-progress, February, 1997. http://www.internic.net/internet-drafts/draft-leach-uuids-guids-00.txt

[Maloney, 1996] M. Maloney, "Hypertext Links in HTML." Internet draft (expired), work-in-progress, January, 1996.

[MARC, 1994] Network Development and MARC Standards, Office, ed. 1994. "USMARC Format for Bibliographic Data", 1994. Washington, DC: Cataloging Distribution Service, Library of Congress.

[Miller et al., 1996] J. Miller, T. Krauskopf, P. Resnick, W. Treese, "PICS Label Distribution Label Syntax and Communication Protocols" Version 1.1, W3C Recommendation REC-PICS-labels-961031. http://www.w3.org/pub/WWW/TR/REC-PICS-labels-961031.html.

[Slein et al., 1997] J. A. Slein, F. Vitali, E. J. Whitehead, Jr., D. Durand, "Requirements for Distributed Authoring and Versioning Protocol for the World Wide Web." RFC XXXX. Xerox, Univ. of Bologna, U.C. Irvine, Boston Univ. YYY, 1997. ftp://ds.internic.net/rfc/rfcXXXX.txt

[WebDAV, 1997] WEBDAV Design Team. "A Proposal for Web Metadata Operations." Unpublished manuscript. http://www.ics.uci.edu/~ejw/authoring/proposals/metadata.html

[Weibel et al., 1995] S. Weibel, J. Godby, E. Miller, R. Daniel, "OCLC/NCSA Metadata Workshop Report." http://purl.oclc.org/metadata/dublin_core_report.

[Yergeau, 1997] F. Yergeau, "UTF-8, a transformation format of Unicode and ISO 10646", Internet Draft, work-in-progress, draft-yergeau-utf8-rev-00.txt, http://www.internic.net/internet-drafts/draft-yergeau-utf8-rev-00.txt.

21. Authors' Addresses

Y. Y. Goland
Microsoft Corporation
One Microsoft Way
Redmond, WA 98052-6399
Email: yarong@microsoft.com

E. J. Whitehead, Jr.
Dept. Of Information and Computer Science
University of California, Irvine
Irvine, CA 92697-3425
Email: ejw@ics.uci.edu

A. Faizi
Netscape
685 East Middlefield Road
Mountain View, CA 94043
Email: asad@netscape.com

S. R. Carter
Novell
1555 N. Technology Way
M/S ORM F111
Orem, UT 84097-2399
Email: srcarter@novell.com

D. Jensen
Novell
1555 N. Technology Way
M/S ORM F111
Orem, UT 84097-2399
Email: dcjensen@novell.com