WEBDAV Working Group
INTERNET DRAFT
<draft-ietf-webdav-protocol-06>

Expires July, 1998

Y.Y. Goland, Microsoft
E.J. Whitehead, Jr., U.C. Irvine
A. Faizi, Netscape
S.R. Carter, Novell
D. Jensen, Novell
January 18, 1998

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

Status of this Memo

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

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

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, and resource locking (collision avoidance).

Changes

Changes since draft-ietf-webdav-protocol-06.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.]

Rationale for many of the changes made in this revision of the draft can be found in the mailing list archives at:
http://lists.w3.org/Archives/Public/w3c-dist-auth/1997OctDec/0160.html.

Where the 200 OK status code was used to indicate a successful response without a response entity body, 204 No Content is now used.
Because PEP uses 420 and 421 status codes, and since PEP has been submitted as an Experimental RFC, the WebDAV 420 status code has been changed to 422, and the WebDAV 421 status code has been changed to 423.

The 423 Destination Locked status code has been changed to 423 Locked, and now covers all cases where an operand was locked, preventing the execution of the method.

Removed the Destroy header, since it is not needed in this draft, but will be needed in the versioning draft.

The Enforce-Live-Properties header was renamed to Property-Behavior, to more closely represent the meaning of the header now that the "omit" functionality is included. A keepalive field was added to the Property-Behavior header to make it more meaningful.

Removed the INDEX method, since the functionality of INDEX can now be performed by the PROPFIND method. PROPFIND provides more flexibility in specifying the type and amount of property information returned than does INDEX, which is important for returning information on a large number of resources.

Clarified that performing a MOVE as a COPY, then DELETE, performed atomically, only applies to non-collection resources.

Clarified the semantics of errors that are encountered in infinite depth move and copy of a hierarchy of resources. For errors copying internal nodes of the hierarchy tree (i.e., collections), the operation skips that subtree, and moves on to the next subtree. If an error is encountered moving/copying a leaf of the tree, then skip that resource, and move on to the next leaf.

Removed the PATCH method. This will be resubmitted as the document draft-ietf-webdav-patch-00.

Added language that states that if a PROPPATCH is invoked on a null resource (e.g., a deleted resource), an empty resource is created, and the PROPPATCH directives are performed on this new resource.

Added a forward reference to the source link definition (Section 13.11) in Section 4.4.

Changed all Values= to Values:. Also changed all "values" to "value".

References to state tokens are now restricted to sections 9.7 and 9.8.

The property-behavior header has been turned into the propertybehavior XML element because it contained a list of URIs which can thus have unbounded size. The lock-info header has been turned into the lockinfo XML element for the same reason. I have also made the same change of the Propfind header into the Propfind XML element. We can put the property behavior header into the body because neither COPY nor MOVE have bodies. However we can't put lock-token, if-state-match, etc. in the body because they may need to be used with PUT. However I don't consider this a big deal because I sincerely doubt that there will be cases where lock-token or if-state-match will see large numbers of entries.

Also changed omit to mean "copy properties with best effort but failure is acceptable."

Added the external members property.

Added language to 6.4 making it clear that any new resources created as the child of a write locked collection is added to the lock.

Made the lock-token response header from a single URL to multiple URLs. But all the URLs MUST refer to the exact same lock.

<?XML version="1.0"> changed to the correct form: <?xml version="1.0"?>

Changed the delete rule for collections to read that if a delete in a collection member fails then it is the ancestors, not the progeny, who can not be deleted in order to maintain the namespace.

Updated our reference to the XML spec.

Added LOCK and UNLOCK to the list of methods covered by the write lock. This is necessary so that a lock-token will have to be submitted in order to make changes, otherwise we defeat the whole purpose of requiring the lock-token.

Changed the title of section 6.6 from Re-Issuing Write Locks to Refreshing Write Locks, made it illegal to make the same lock request twice (you know you are making the same request because you had to include the lock-token to make it!) and instead made it legal to submit a LOCK method with no body but with a lock-token header. I also added a refresh example.

Put in a note that an empty request body for PROPFIND means to return all names and values of properties on the resources.

I have added a section on XML processing errors. I know, I know, it shouldn't be in the standard. I will move it to our compliance draft as soon as we prepare the first version.

Removed addlocks and replaced with the depth header and the depth element.

Changed all the as in namespace elements to all lower case.

Moved all XML element declarations to the same section. Removed the parent description.

Updated the depth section to make it more generic, changed the wording for how COPY/MOVE are handled with write locks, require that ALL propfind responses include href, require that if a property is not found in a propfind then a 404 Not Found must be returned, and made explicit that PROPFIND responses on resources with internal members are returned as a flat list with no significance to its ordering.

Removed reference to efficient update in the introduction since PATCH is now gone.

Rewrote the write lock and null resource section to deal with the question of the state of the resource when it is locked and null.

Changed www.ietf.org to www.iana.org.

Changed the response element and added the new propstat element. With the prohibition that an HREF can only appear once in a multistatus response we can guarantee linear processing costs.

Added Intellectual Property section, as required by RFC 2026.

Added IANA Considerations section.

Added Authorization headers to LOCK and UNLOCK examples.

Changed lock tokens in examples to use string format of UUID.

Since the latest HTTP revision defines a 418 and 419 status code, the 418 status code has been changed to 422, 419 to 423, 422 to 424, and 423 to 425.

Changed implementation of the get* (e.g., getcontentlength) properties to strength MUST.

Changed definition of XML elements and DAV properties to use XML element definitions, rather than BNF.

Renumbered all sections

Contents

1 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 their authors, creation dates, 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 changes without merging the other author's changes

Namespace Operations: The ability to copy and move Web resources

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 2), collections of resources (Section 3), and locking operations (Section 4). These sections introduce the abstractions manipulated by the WebDAV-specific HTTP methods described in Section 7, "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, Paoli, Sperberg-McQueen, 1998] 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 HTTP headers. Section 8 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 11.

While the status codes provided by HTTP/1.1 are sufficient to describe most 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 9. Since some WebDAV methods may operate over many resources, the Multi-Status status type has been introduced to return status information for multiple resources. Multi-Status response is described in Section 10.

WebDAV employs the property mechanism 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 12 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 13), on internationalization support (Section 14), and on security (Section 15).

2 Data Model for Resource Properties

2.1 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 "dead". A live property has its syntax and semantics enforced by the server. Live properties include cases where 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 the server performs syntax checking on submitted values. A dead property has its syntax and semantics enforced by the client; the server merely records the value of the property verbatim.

2.2 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, XML [Bray, Paoli, Sperberg-McQueen, 1998], Web Collections, and several proposals on representing relationships within HTML. Work on PICS-NG and Web Collections has been subsumed by the Resource Definition Framework (RDF) metadata activity of the World Wide Web Consortium. RDF 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.

2.3 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 set or retrieve just those properties.

2.4 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 when they encounter extensions 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 any human-readable property to be encoded and read in a character set familiar to the user.

2.5 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 prevents namespace collisions and provides 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.

3 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 Uniform Resource Locator (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.

3.1 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, the internal member's URI is equal to the parent collection's URI plus an additional segment where segment is defined in Section 3.2.1 of RFC 2068 [Fielding et al., 1996].

An external member resource is a resource that could not be an internal member resource. Any given internal or external Member 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 invokes a method 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. In general clients SHOULD use the "/" form of collection names.

3.2 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.

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

3.3 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.

3.4 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.iana.org/standards/dav/source (see Section 12.11 for a description of the source link). 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.

4 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. This document defines locking for only one access type, write. However, the syntax is extensible, and permits the eventual specification of locking for other access types.

4.1 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 may be 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 may be 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.

4.2 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.

4.3 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.

4.4 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-[None]-State-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 Universally Unique Identifier (UUID, also known as a Globally Unique Identifier, or GUID) mechanism, as described in [Leach, Salz, 1998].

Opaquelocktoken generators, however, have a choice of how they create these tokens. They can either generate a new UUID for every lock token they create, which is potentially very expensive, or they can create a single UUID 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 UUID.

```
OpaqueLockToken-URI = "opaquelocktoken:" UUID [Extension]  ; The UUID
production is the string form of a UUID, as defined in [Leach, Salz, 1998].
Note that white space (LWS) is not allowed between elements of this
production.
```

Extension = path ; path is defined in Section 3.2.1 of RFC 2068 [Fielding et al., 1996]

4.5 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.

4.6 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.

5 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.

5.1 Methods Restricted by Write Locks

A write lock prevents a principal without the lock from successfully executing a PUT, POST, PROPPATCH, LOCK, UNLOCK, 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.

5.2 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.

5.3 Write Locks and Null Resources

It is possible to assert a write lock on a null resource in order to lock the name. A write locked null resource acts in all ways as a null resource other than it MUST respond to a PROPFIND request and MUST support the lockdiscovery and supportedlock properties.

Until a method such as PUT or MKCOL is executed, the resource stays in the null state with the exception of the behavior stated above.

If the resource is unlocked without a PUT, MKCOL, or similar method having been executed, the resource is no longer required to support the PROPFIND method or the lockdiscovery and supportedlock properties.

5.4 Write Locks and Collections

A write lock on a collection prevents the addition or removal of members of the collection by non-lock owners. As a consequence, when a principal issues a request to create a new internal member of a write locked collection using PUT or POST, or to remove an existing internal member of a write locked 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 425 Locked status code.

If a lock owner causes a resource to be added as an internal member of a locked collection then the new resource is automatically added to the lock. This is the only mechanism that allows a resource to be added to a write lock. Thus, for example, if the collection /a/b/ is write locked and the resource /c is moved to /a/b/c then /a/b/c will be added to the write lock.

5.5 Write Locks and COPY/MOVE

A COPY method invocation MUST NOT duplicate any write locks active on the source. However, as previously noted, if the COPY copies the resource into a collection that is depth locked then the resource will be added to the lock.

A MOVE does not move the write lock with the resource. There are two exceptions to this rule. First, as noted in section 5.4, if the MOVE makes the resource a child of a collection that is depth locked then the resource will be under the same lock. Second, if a depth locked resource is moved to a destination that is within the scope of the same depth lock (e.g., within the namespace tree covered by the lock), the moved resource is still a member of the lock. In both cases a Lock-Token header MUST be submitted containing a lock token for the lock on the source, if locked, and on the destination.

5.6 Refreshing Write Locks

A client MUST NOT submit the same write lock request twice. Note that a client is always aware it is resubmitting the same lock request because it must include the Lock-Token header in order to make the request for a resource that is already locked.

However, a client MAY submit a LOCK method with a Lock-Token header but without a body. This form of LOCK MAY only be used to "refresh" a lock. Currently, refreshing a lock only means that any timers associated with the lock are re-set.

A server MAY return a Timeout header with a lock refresh that is different than the Timeout header returned when the lock was originally requested. Additionally clients MAY submit Timeout headers of arbitrary value with their lock refresh requests. Servers, as always, MAY ignore Timeout headers submitted by the client.

If an error is received in response to a refresh LOCK request the client MUST assume that the lock was not refreshed.

5.7 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, defined in Section 8.7, is introduced.

5.7.1 Write Lock Token Example

>>Request

```
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:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>
Authorization: Digest username="fielding",
   realm="fielding@ics.uci.edu", nonce="...",
   uri="/~fielding/index.html", response="...",
   opaque="..."
```

>>Response

```
HTTP/1.1 204 No Content
```

In this example, even though both the source and destination are locked, only one lock token must be submitted, for the lock on the destination. This is due to the source resource not being modified during a COPY, and hence unaffected by the write lock. The Authorization header provides the Digest authentication credentials for the principal making the request (note that the nonce, response, and opaque fields have not been calculated for this example). The source and the destination resources are both located within the same authentication realm, therefore only one set of Authorization credentials needs to be submitted.

6 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 [Fielding et al., 1997]. Since this augmented BNF uses the basic production rules provided in Section 2.2 of [Fielding et al., 1997], 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].

7 HTTP Methods for Distributed Authoring

7.1 PROPFIND

The PROPFIND method retrieves properties defined on the Request-URI, if the resource does not have any internal members, or on the Request-URI and potentially its member resources, if the resource does have internal members. All DAV compliant resources MUST support the PROPFIND method.

A client MAY submit a Depth header with a value of "0", "1", or "infinity" with a PROPFIND on a resource with internal members. DAV compliant servers MUST support the "0", "1" and "infinity" behaviors. By default, the PROPFIND method without a Depth header MUST act as if a "Depth: infinity" header was included.

A client MAY submit a propfind XML element in the body of the request method 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. A client MAY choose not to submit a request body. An empty request body MUST be treated as a request for the names and values of all 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 there is an error retrieving a property then a proper error result must be included. Requests to retrieve the value of a property which does not exist is an error and MUST be noted with a response XML element which contains a 404 Not Found status value.

Consequently, the multistatus XML element for a resource with members MUST include a response XML element for each member of the resource, to whatever depth was requested. Each response XML element MUST contain an href XML element that identifies the resource on which the properties in the prop XML element are defined. Results for a PROPFIND on a resource with internal members are returned as a flat list whose order of entries is not significant.

In the case of allprop and propname, if a principal does not have the right to know if a particular property exists then a 404 Not Found MUST be returned.

The results of this method SHOULD NOT be cached.

7.1.1 Example: Retrieving Named Properties

>>Request

```
PROPFIND  /files/ HTTP/1.1
Host: www.foo.bar
Depth: 0
Content-type: text/xml
Content-Length: xyz

<?xml version="1.0"?>
<?namespace href ="http://www.iana.org/standards/dav/" as="D"?>
<D:propfind>
       <D:href>http://www.foo.bar/boxschema/bigbox</D:href>
```

```
        <D:href>http://www.foo.bar/boxschema/author</D:href>
        <D:href>http://www.foo.bar/boxschema/DingALing</D:href>
        <D:href>http://www.foo.bar/boxschema/Random</D:href>
</D:propfind>
```

>>Response

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

<?xml version="1.0"?>
<?namespace href ="http://www.iana.org/standards/dav/" as="D"?>
<?namespace href="http://www.foo.bar/boxschema" as="R"?>
<D:multistatus>
        <D:response>
                <D:href>http://www.foo.bar/files/</D:href>
                <D:propstat>
                        <D:prop>
                                <R:bigbox>
                                        <R:BoxType>Box type A</R:BoxType>
                                </R:bigbox>
                                <R:author>
                                        <R:Name>J.J. Johnson</R:Name>
                                </R:author>
                        </D:prop>
                        <D:status>HTTP/1.1 200 OK</D:status>
                </D:propstat>
                <D:propstat>
                        <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:propstat>
        </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 XML element 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.

7.1.2 Example: Using allprop to Retrieve All Properties

>>Request

```
PROPFIND  /container/ HTTP/1.1
Host: www.foo.bar
Depth: 1
Content-Type: text/xml
Content-Length: xxxxx

<?xml version="1.0"?>
<?namespace href ="http://www.iana.org/standards/dav/" as="D"?>
<D:propfind>
        <D:allprop/>
</D:propfind>
```

>>Response

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

<?xml version="1.0"?>
<?namespace href="http://www.iana.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:propstat>
                     <S:prop>
                            <R:bigbox>
                                   <R:BoxType>Box type A</R:BoxType>
                            </R:bigbox>
                            <R:author>
                                   <R:Name>Hadrian</R:Name>
                            </R:author>
                <S:creationdate>
                  1997-12-01T17:42:21-08:00
                </S:creationdate>
                <S:displayname>
                  Example collection
                </S:displayname>
                <S:externalmembers>
                  <S:href>http://www.acme.com/front/</S:href>

                </S:externalmembers>
                <S:resourcetype><S:collection/></S:resourcetype>
                <S:supportedlock>
                  <S:lockentry>
                    <S:exclusive/><S:write/>
                  </S:lockentry>
                  <S:lockentry>
                    <S:shared/><S:write/>
                  </S:lockentry>
                </S:supportedlock>
                    </S:prop>
                    <S:status>HTTP 1.1 200 OK</S:status>
              </S:propstat>
       </S:response>
       <S:response>
              <S:href>http://www.foo.bar/container/front.html</S:href>
              <S:propstat>
                     <S:prop>
                            <R:bigbox>
                                   <R:BoxType>Box type B</R:BoxType>
                            </R:bigbox>
                <S:creationdate>
                  1997-12-01T18:27:21-08:00
                </S:creationdate>
                <S:displayname>
                  Example HTML resource
                </S:displayname>
                <S:getcontentlength>
                  4525
                </S:getcontentlength>
                <S:getcontenttype>
                  text/html
                </S:getcontenttype>
                <S:getetag>
```

```
                     zzyzx
                      </S:getetag>
                   <S:getlastmodified>
                     Monday, 12-Jan-98 09:25:56 GMT
                   </S:getlastmodified>
                   <S:resourcetype/>
                   <S:supportedlock>
                     <S:lockentry>
                       <S:exclusive/><S:write/>
                     </S:lockentry>
                     <S:lockentry>
                       <S:shared/><S:write/>
                     </S:lockentry>
                   </S:supportedlock>
                       </S:prop>
                       <S:status>HTTP 1.1 200 OK</S:status>
                </S:propstat>
         </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 XML element containing the allprop XML element, meaning the request should return the name and value of all properties defined on each resource.

The resource http://www.foo.bar/container/ has seven properties defined on it:

http://www.foo.bar/boxschema/bigbox, http://www.foo.bar/boxschema/author, http://www.iana.org/standards/dav/creationdate, http://www.iana.org/standards/dav/displayname, http://www.iana.org/standards/dav/externalmembers, http://www.iana.org/standards/dav/resourcetype, and http://www.iana.org/standards/dav/supportedlock.

The last five properties are WebDAV-specific, defined in Section 12. Since GET is not supported on this resource, the get-* properties (e.g., get-content-length) are not defined on this resource. The DAV-specific properties assert that "container" was created on December 1, 1997, at 5:42:21PM, in a time zone 8 hours west of GMT (creationdate), has a name of "Example collection" (displayname), a single external member resource, http://www.acme.com/front/ (externalmembers), a collection resource type (resourcetype), and supports exclusive write and shared write locks (supportedlock).

The resource http://www.foo.bar/container/front.html has nine properties defined on it:

http://www.foo.bar/boxschema/bigbox (another instance of the "bigbox" property type), http://www.iana.org/standards/dav/creationdate, http://www.iana.org/standards/dav/displayname, http://www.iana.org/standards/dav/getcontentlength, http://www.iana.org/standards/dav/getcontenttype, http://www.iana.org/standards/dav/getetag, http://www.iana.org/standards/dav/getlastmodified, http://www.iana.org/standards/dav/resourcetype, and http://www.iana.org/standards/dav/supportedlock.

The DAV-specific properties assert that "front.html" was created on December 1, 1997, at 6:27:21PM, in a time zone 8 hours west of GMT (creationdate), has a name of "Example HTML resource" (displayname), a content length of 4525 (getcontentlength), a MIME type of "text/html" (getcontenttype), an entity tag of "zzyzx" (getetag), was last modified on Monday, January 12, 1998, at 09:25:56 GMT (getlastmodified), has an undefined resource type, meaning that it is not a collection (resourcetype), and supports both exclusive write and shared write locks (supportedlock).

7.1.3 Example: Using propname to Retrieve all Property Names

>>Request

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

   <?xml version="1.0"?>
   <?namespace href ="http://www.iana.org/standards/dav/" as="D"?>
   <D:propfind>
          <D:propname/>
   </D:propfind>
```

>>Response

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

   <?xml version="1.0"?>
   <?namespace href="http://www.iana.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:propstat>
                        <D:prop>
                               <R:bigbox/>
                               <R:author/>
                    <D:creationdate/>
                    <D:displayname/>
                    <D:externalmembers/>
                    <D:resourcetype/>
                    <D:supportedlock/>
                        </D:prop>
                        <D:status>HTTP 1.1 200 OK</D:status>
                 </D:propstat>
          </D:response>
          <D:response>
                 <D:href>http://www.foo.bar/container/front.html</D:href>
                 <D:propstat>
                        <D:prop>
                               <R:bigbox/>
                    <D:creationdate/>
                    <D:displayname/>
                    <D:get-content-length/>
                    <D:get-content-type/>
                    <D:get-etag/>
                    <D:get-last-modified/>
                    <D:resourcetype/>
                    <D:supportedlock/>
                        </D:prop>
                        <D:status>HTTP 1.1 200 OK</D:status>
                 </D:propstat>
          </D:response>
   </D:multistatus>
```

In this example, PROPFIND is invoked on the collection resource http://www.foo.bar/container/, with a propfind XML element containing the propname XML element, 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 seven properties defined on it, http://www.foo.bar/boxschema/bigbox, and http://www.foo.bar/boxschema/author, http://www.iana.org/standards/dav/creationdate, http://www.iana.org/standards/dav/displayname, http://www.iana.org/standards/dav/externalmembers, http://www.iana.org/standards/dav/resourcetype, and http://www.iana.org/standards/dav/supportedlock. The resource http://www.foo.bar/container/index.html, a member of the "container" collection, has nine properties defined on it, http://www.foo.bar/boxschema/bigbox, http://www.iana.org/standards/dav/creationdate, http://www.iana.org/standards/dav/displayname, http://www.iana.org/standards/dav/get-content-length, http://www.iana.org/standards/dav/get-content-type, http://www.iana.org/standards/dav/get-etag, http://www.iana.org/standards/dav/get-last-modified, http://www.iana.org/standards/dav/resourcetype, and http://www.iana.org/standards/dav/supportedlock.

7.2 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 SHOULD 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). Instructions MUST either all be executed or none executed. Thus if any error occurs during processing all executed instructions MUST be undone and a proper error result returned. Instruction processing details can be found in the definition of the set and remove instructions in Section 11.13.

If PROPPATCH is invoked on a null resource (e.g., a deleted resource), an empty resource is created, and the PROPPATCH directives are performed on this new resource.

7.2.1 Status Codes

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

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

409 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.

7.2.2 Example

>>Request

```
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.iana.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>
```

>>Response

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

   <?xml version="1.0"?>
   <?namespace href="http://www.iana.org/standards/dav/" as="D"?>
   <?namespace href="http://www.w3.com/standards/z39.50/" as="Z"?>
   <D:multistatus>
          <D:response>
                 <D:href>http://www.foo.com/bar</D:href>
                 <D:propstat>
                        <D:prop><Z:Authors/></D:prop>
                        <D:status>HTTP/1.1 424 Method Failure</D:status>
                 </D:propstat>
                 <D:propstat>
                        <D:prop><Z:Copyright-Owner/></D:prop>
                        <D:status>HTTP/1.1 409 Conflict</D:status>
                 </D:propstat>
                 <D:responsedescription> Copyright Owner can not be deleted or
   altered.</D:responsedescription>
          <D:response>
   </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 status code for the Authors property indicates this action would have succeeded if it were not for the conflict with removing the Copyright-Owner property.

7.3 MKCOL Method

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

7.3.1 Request

MKCOL creates a new collection resource at the location specified by the Request-URI. If the resource identified by the Request-URI is non-null then the 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.

7.3.2 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.

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

7.3.3 Example

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

>>Request

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

>>Response

```
HTTP/1.1 201 Created
```

7.4 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.

7.4.1 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.

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.

More than one Collection-Member request header MUST NOT be used with the ADDREF method.

7.4.2 Example

>>Request

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

>>Response

```
HTTP/1.1 204 No Content
```

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

7.5 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.

7.5.1 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.

More than one Collection-Member request header MUST NOT be used with the DELREF method.

7.5.2 Example

>>Request

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

>>Response

```
HTTP/1.1 204 No Content
```

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

7.6 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. Hence it is possible that 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.

7.7 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.

7.8 DELETE

7.8.1 DELETE 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.

7.8.2 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 ancestors 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.

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.

7.8.2.1 Example

>>Request

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

```
>>Response

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

<?xml version="1.0"?>
<?namespace href="http://www.iana.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/1.1 424 Method Failure</d:status>
       </d:response>
       <d:response>
              <d:href>http://www.foo.bar/container/resource3</d:href>
              <d:status>HTTP/1.1 425 Locked</d:status>
       </d:response>
</d:multistatus>
```

In this example the attempt to delete http://www.foo.bar/container/resource3 failed because it is locked, and no lock token was submitted with the request. 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.

7.9 PUT

7.9.1 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 but are not otherwise effected. 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 409 Conflict.

7.9.2 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.

7.10 COPY Method

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.

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.

7.10.1 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. Subsequent alterations to the destination resource will not modify the source resource. Subsequent alterations to the source resource will 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 and XML elements, following the definition for copying properties.

7.10.2 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. If a property cannot be copied live, then its value MUST be duplicated, octet-for-octet, in an identically named, dead property on the destination resource.

The propertybehavior XML element can specify that properties are copied on best effort, that all live properties MUST be successfully copied or the method MUST fail, or that a specified list of live properties MUST be successfully copied or the method must fail. The propertybehavior XML element is defined in Section 11.12.

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 412 Precondition Failed.

7.10.3 COPY for Collections

The COPY method on a collection without a Depth header MUST act as if a Depth header with value "infinity" 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 and the records of its external member resources is to be copied to the location specified in the Destination header, and all its internal member resources are to be copied to a location relative to it, recursively through all levels of the collection hierarchy.

A COPY of depth "0" only instructs that the collection, the properties, and the records of 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. However, if an error occurs while copying an internal member collection, all members of this collection MUST NOT be copied. In this case, after detecting the error, the COPY operation SHOULD try to finish as much of the original copy operation as possible. So, for example, if an infinite depth copy operation is performed on collection /a/, which contains collections /a/b/ and /a/c/, and an error occurs copying /a/b/, an attempt should still be made to copy /a/c/. Similarly, after encountering an error copying a non-collection resource as part of an infinite depth copy, the server SHOULD try to finish as much of the original copy operation as possible.

The response is a Multi-Status status code with an entity body that describes the result of the COPY on each affected resource. The href XML element in the response refers to 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 XML element 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 425 Locked error.

7.10.4 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.

7.10.5 Status Codes

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

204 No Content - The source resource was successfully copied to a pre-existing destination resource. Since there is no entity body in the response, 204 No Content is used instead of 200 OK.

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

423 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.

425 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 on another server and the destination server refuses to accept the resource.

7.10.6 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 204 No Content status code indicates the existing resource at the destination was overwritten.

>>Request

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

```
   Destination: http://www.ics.uci.edu/users/f/fielding/index.html
```

>>Response

```
   HTTP/1.1 204 No Content
```

7.10.7 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.

>>Request

```
   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
```

>>Response

```
   HTTP/1.1 412 Precondition Failed
```

7.10.8 Collection Example

>>Request

```
   COPY /container/ HTTP/1.1
   Host: www.foo.bar
   Destination: http://www.foo.bar/othercontainer/
   Depth: infinity
   Content-Type: text/xml
   Content-Length: xxxxx

   <?xml version="1.0"?>
   <?namespace href="http://www.iana.org/standards/dav/" as="d"?>
   <d:propertybehavior>
         <d:keepalive>*</d:keepalive>
   </d:propertybehavior>
```

>>Response

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

   <?xml version="1.0"?>
   <?namespace href="http://www.iana.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:status>HTTP/1.1 201 Created</d:status>
         </d:response>

         <d:response>
           <d:href>http://www.foo.bar/othercontainer/R2/</d:href>
           <d:href>http://www.foo.bar/othercontainer/R2/D2</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 maintaining the liveness of properties (this is specified by the propertybehavior XML element). Since an error occurred copying R2, R2's member D2 was not copied.

7.11 MOVE Method

The MOVE operation on a non-collection resource is the logical equivalent of a copy (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.

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.

7.11.1 MOVE for Collections

A MOVE of depth infinity instructs that the collection specified in the Request-URI, including the records of its external member resources, is to be moved to the location specified in the Destination header, and all its internal member resources are to be moved to locations relative to it, recursively through all levels of the collection hierarchy.

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. However, if an error occurs while moving an internal member collection, all members of the failed collection MUST NOT be moved. In this case, after detecting the error, the move operation SHOULD try to finish as much of the original move as possible. So, for example, if an infinite depth move is performed on collection /a/, which contains collections /a/b/ and /a/c/, and an error occurs moving /a/b/, an attempt should still be made to try moving /a/c/. Similarly, after encountering an error moving a non-collection resource as part of an infinite depth move, the server SHOULD try to finish as much of the original move operation as possible.

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 affected resource. The href XML element in the response refers to 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 425 Locked error.

7.11.2 Status Codes

201 Created - The source resource was successfully moved, and a new resource was created at the destination.

204 No Content - The move operation was successful, and the resource at the destination was overwritten.

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

425 Locked - The source or 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 on another server and the destination server refuses to accept the resource.

7.11.3 Non-Collection 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 would have been overwritten if the destination resource had been non-null. In this case, since there was nothing at the destination resource, the response code is 201 Created.

>>Request

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

>>Response

```
HTTP/1.1 201 Created
Location: http://www.ics.uci.edu/users/f/fielding/index.html
```

7.11.4 Collection Example

>>Request

```
MOVE /container/ HTTP/1.1
Host: www.foo.bar
Destination: http://www.foo.bar/othercontainer/
Overwrite: F
Lock-Token: <opaquelocktoken:fe184f2e-6eec-41d0-c765-01adc56e6bb4>,
   <opaquelocktoken:e454f3f3-acdc-452a-56c7-00a5c91e4b77>
Content-Type: text/xml
Content-Length: xyz
Authorization: Digest username="rohit",
   realm="rohit@www.foo.bar", nonce="...",
   uri="/container/", response="...",
   opaque="..."

<?xml version="1.0"?>
```

```
<?namespace href="http://www.iana.org/standards/dav/" as="d"?>
<d:propertybehavior>
      <d:keepalive>*</d:keepalive>
</d:propertybehavior>
```

>>Response

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

<?xml version="1.0"?>
<?namespace href="http://www.iana.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:status>HTTP/1.1 204 No Content</d:status>
      </d:response>
      <d:response>
        <d:href>http://www.foo.bar/container/C2/R2</d:href>
        <d:status>HTTP/1.1 424 Method Failure</d:status>
      <d:response>
        <d:href>http://www.foo.bar/othercontainer/C2/</d:href>
        <d:status>HTTP/1.1 425 Locked</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 /container/C2/ could not be moved. As the attempt to move /container/C2/ failed then the resource /container/C2/R2 MUST also fail since it is a child of /container/C2/.

7.12 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.

7.12.1 Operation

A LOCK method invocation creates the lock specified by the lockinfo XML element on the Request-URI. Lock method requests SHOULD have a XML request body which contains an owner XML element for this lock request, unless this is a refresh 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.

7.12.2 The Effect of Locks on Properties and Collections

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.

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.

7.12.3 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.

7.12.4 Depth and Locking

The Depth header MAY be used with the LOCK method. Values other than 0 or infinity MUST NOT be used with the Depth header.

A Depth header of value 0 means to just lock the resource specified by the request-URI.

If the Depth header is set to infinity then the resource specified in the request-URI along with all its internal members, all the way down the hierarchy, are to be locked. 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. Hence, partial success is not an option. Either the entire hierarchy is locked or no resources are locked.

7.12.5 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.

7.12.6 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 MUST 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.

7.12.7 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.

7.12.8 Status Codes

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

425 Locked - The resource is locked, so the method has been rejected.

7.12.9 Example - Simple Lock Request

>>Request

```
LOCK /workspace/webdav/proposal.doc HTTP/1.1
Host: webdav.sb.aol.com
Timeout: Infinite, Second-4100000000
Content-Type: text/xml
Content-Length: xyz
Authorization: Digest username="ejw",
   realm="ejw@webdav.sb.aol.com", nonce="...",
   uri="/workspace/webdav/proposal.doc",
   response="...", opaque="..."


<?xml version="1.0"?>
<?namespace href="http://www.iana.org/standards/dav/" as="D"?>
<D:lockinfo>
       <D:locktype><D:write/></D:locktype>
       <D:lockscope><D:exclusive/></D:lockscope>
       <D:owner>
         <D:href>http://www.ics.uci.edu/~ejw/contact.html</D:href>
       </D:owner>
</D:lockinfo>
```

>>Response

```
HTTP/1.1 200 OK
Lock-Token: <opaquelocktoken:e71d4fae-5dec-22d6-fea5-00a0c91e6be4>
Timeout: Second-604800
Content-Type: text/xml
Content-Length: xxxxx

<?xml version="1.0"?>
<?namespace href ="http://www.iana.org/standards/dav/" as="D"?>
<D:prop>
       <D:lockdiscovery>
         <D:activelock>
           <D:locktype><D:write/></D:locktype>
           <D:lockscope><D:exclusive/></D:lockscope>
           <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:e71d4fae-5dec-22d6-fea5-00a0c91e6be4
               </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. Note that the nonce, response, and opaque fields have not been calculated in the Authorization request header.

7.12.10 Example - Refreshing a Write Lock

>>Request

```
LOCK /workspace/webdav/proposal.doc HTTP/1.1
Host: webdav.sb.aol.com
Timeout: Infinite, Second-4100000000
Lock-Token: <opaquelocktoken:e71d4fae-5dec-22d6-fea5-00a0c91e6be4>
Authorization: Digest username="ejw",
   realm="ejw@webdav.sb.aol.com", nonce="...",
   uri="/workspace/webdav/proposal.doc",
   response="...", opaque="..."
```

>>Response

```
HTTP/1.1 200 OK
Lock-Token: <opaquelocktoken:e71d4fae-5dec-22d6-fea5-00a0c91e6be4>
Timeout: Second-604800
Content-Type: text/xml
Content-Length: xxxxx

<?xml version="1.0"?>
<?namespace href ="http://www.iana.org/standards/dav/" as="D"?>
<D:prop>
      <D:lockdiscovery>
        <D:activelock>
          <D:locktype><D:write/></D:locktype>
          <D:lockscope><D:exclusive/></D:lockscope>
          <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:e71d4fae-5dec-22d6-fea5-00a0c91e6be4
              </D:href>
          </D:locktoken>
        </D:activelock>
      </D:lockdiscovery>
</D:prop>
```

This request would refresh the lock, resetting any time outs. Notice that the client asked for an infinite time out but the server choose to ignore the request. In this example, the nonce, response, and opaque fields have not been calculated in the Authorization request header.

7.12.11 Example - Multi-Resource Lock Request

>>Request

```
LOCK /webdav/ HTTP/1.1
Host: webdav.sb.aol.com
Timeout: Infinite, Second-4100000000
Depth: infinity
Authorization: Digest username="ejw",
   realm="ejw@webdav.sb.aol.com", nonce="...",
   uri="/workspace/webdav/proposal.doc",
   response="...", opaque="..."

<?xml version="1.0"?>
<?namespace href="http://www.iana.org/standards/dav/" as="D"?>
<D:lockinfo>
      <D:locktype><D:write/></D:locktype>
      <D:lockscope><D:exclusive/></D:lockscope>
      <D:owner>
        <D:href>http://www.ics.uci.edu/~ejw/contact.html</D:href>
      </D:owner>
</D:lockinfo>
```

>>Response

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

<?xml version="1.0"?>
<?namespace href="http://www.iana.org/standards/dav/" as="D"?>
<D:multistatus>
      <D:response>
            <D:href>http://webdav.sb.aol.com/webdav/proposal.doc</D:href>
            <D:href>http://webdav.sb.aol.com/webdav/</D:href>
            <D:status>HTTP/1.1 424 Method Failure</D:status>
      </D:response>
      <D:response>
            <D:href>http://webdav.sb.aol.com/webdav/secret</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 a collection and all its children. 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 request entity body contains the contact information for the principal taking out the lock, in this case a web page URL.

The 424 Method Failure indicates that a lock was not taken out on these resources due to an error elsewhere. Note that this does not mean that a lock would have succeeded on these resources had the other error not occurred. It only means that another error has occurred and so the entire method has been aborted. The error is a 403 Forbidden response on the resource http://webdav.sb.aol.com/webdav/secret. Because this resource could not be locked, none of the resources were locked.

In this example, the nonce, response, and opaque fields have not been calculated in the Authorization request header.

7.13 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.

7.13.1 Example

>>Request

```
UNLOCK /workspace/webdav/info.doc HTTP/1.1
Host: webdav.sb.aol.com
Lock-Token:<opaquelocktoken:a515cfa4-5da4-22e1-f5b5-00a0451e6bf7>
Authorization: Digest username="ejw",
   realm="ejw@webdav.sb.aol.com", nonce="...",
   uri="/workspace/webdav/proposal.doc",
   response="...", opaque="..."
```

>>Response

```
HTTP/1.1 204 No Content
```

In this example, the lock identified by the lock token "opaquelocktoken:a515cfa4-5da4-22e1-f5b5-00a0451e6bf7" 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 is removed from all resources included in the lock. The 204 status code is used instead of 200 OK because there is no response entity body.

In this example, the nonce, response, and opaque fields have not been calculated in the Authorization request header.

8 HTTP Headers for Distributed Authoring

8.1 Collection-Member Header

```
CollectionMember = "Collection-Member" ":" absoluteURI   ; absoluteURI 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.

8.2 DAV Header

```
DAV = "DAV" ":" ["1"] [",2"] ["," 1#extend]
```

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

The value is a list of all compliance classes that the resource supports. Note that above a comma has already been added to the 2. This is because a resource can not be level 2 compliant unless it is also level 1 compliant. Please refer to Section 13 for more details. In general, however, support for one compliance class does not entail support for any other.

8.3 Depth Header

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

The Depth header is used with methods executed on resources which could potentially have internal members to indicate whether the method is to be applied only to the resource (Depth = 0), to the resource and its immediate children, (Depth = 1), or the resource and all its progeny (Depth = infinity).

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 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 = infinity header had been applied.

Clients MUST NOT rely upon methods executing on members of their hierarchies in any particular order or on 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.

The Depth header only specifies the behavior of the method with regards to internal children. If a resource does not have internal children then the Depth header is ignored.

Please note, however, that it is always an error to submit a value for the Depth header that is not allowed by the method's definition. Thus submitting a "Depth: 1" on a COPY, even if the resource does not have internal members, MUST result in a 400 Bad Request. The method should fail not because the resource doesn't have internal members, but because of the illegal value in the header.

8.4 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.

8.5 If-None-State-Match

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

Coded-URL = "<" URI ">"
```

The If-None-State-Match header is intended to have similar functionality to the If-None-Match header defined in section 14.26 of [Fielding et al., 1997]. 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, 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, 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.

8.6 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 [Fielding et al., 1997]. 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 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, the method MUST fail with a 412 Precondition Failed.

8.7 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 given 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 the MOVE method, then the lock tokens, if any, for all affected 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 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.

8.8 Lock-Token Response Header

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

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

If multiple lock-tokens are returned then they MUST all refer to the same lock. As the lock tokens all refer to the same lock a client need only record one of them.

8.9 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.

8.10 Status-URI Response Header

The Status-URI response header MAY be used with the 102 Processing status 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.

8.11 Timeout Header

```
TimeOut = "Timeout" ":" 1#TimeType
TimeType = ("Second-" DAVTimeOutVal | "Infinite" | Other)
```

```
DAVTimeOutVal = 1*digit
Other = Extend field-value   ; See section 4.2 of [Fielding et al., 1997]
```

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 SHOULD be 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. However the lock MUST be refreshed if a refresh LOCK method is successfully received.

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.

9 Status Code Extensions to HTTP/1.1

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

9.1 102 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 status 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.

9.2 207 Multi-Status

The response provides status for multiple independent operations.

9.3 422 Unprocessable Entity

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

9.4 423 Insufficient Space on Resource

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

9.5 424 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 424 Method Failure.

9.6 425 Locked

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

10 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 XML Element Definitions

In the section below, the final line of each section gives the element type declaration using the format defined in [Bray, Paoli, Sperberg-McQueen, 1998]. The "Value" field, where present, specifies futher restrictions on the allowable contents of the XML element using BNF (i.e., to further restrict the values of a PCDATA element).

11.1 activelock XML Element

Name:	activelock
Namespace:	http://www.iana.org/standards/dav/
Purpose:	Describes a lock on a resource.

```
<!ELEMENT activelock (locktype, lockscope, depth?, owner, timeout, locktoken)
>
```

11.1.1 depth XML Element

Name:	depth
Namespace:	http://www.iana.org/standards/dav/
Purpose:	The value of the depth header used to create a lock.

Description: If this element is not included in a lockinfo element then the client MUST assume that the lock is of depth 0.
Value: `"0" | "infinity"`

```
<!ELEMENT depth (#PCDATA) >
```

11.1.2 locktoken XML Element

Name: locktoken
Namespace: http://www.iana.org/standards/dav/
Purpose: The lock token associated with a lock.
Description: The href contains an opaque lock token URI (i.e., the OpaqueLockToken-URI production in Section 4.4).

```
<!ELEMENT locktoken (href) >
```

11.1.3 timeout XML Element

Name: timeout
Namespace: http://www.iana.org/standards/dav/
Purpose: The timeout associated with a lock
Value: `TimeType`

```
<!ELEMENT timeout (#PCDATA) >
```

11.2 collection XML Element

Name: collection
Namespace: http://www.iana.org/standards/dav/
Purpose: Identifies the associated resource as a collection. The resourcetype property of a collection resource MUST have this value.

```
<!ELEMENT collection EMPTY >
```

11.3 href XML Element

Name: href
Namespace: http://www.iana.org/standards/dav/
Purpose: Identifies the content of the element as a URI.
Value: `URI ; See section 3.2.1 of [Fielding et al., 1997]`

```
<!ELEMENT href (#PCDATA)>
```

11.4 link XML Element

Name: link
Namespace: http://www.iana.org/standards/dav/
Purpose: Identifies the property as a link and contains the source and destination of that link.
Description: The link XML element is used to provide the sources and destinations of a link. The name 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.

```
<!ELEMENT link (src+, dst+) >
```

11.4.1 dst XML Element

Name:	dst
Namespace:	http://www.iana.org/standards/dav/
Purpose:	Indicates the destination of a link
Value:	URI

```
<!ELEMENT dst (#PCDATA) >
```

11.4.2 src XML Element

Name:	src
Namespace:	http://www.iana.org/standards/dav/
Purpose:	Indicates the source of a link.
Value:	URI

```
<!ELEMENT src (#PCDATA) >
```

11.5 lockentry XML Element

Name:	lockentry
Namespace:	http://www.iana.org/standards/dav/
Purpose:	Defines the types of locks that can be used with the resource.

```
<!ELEMENT lockentry (lockscope, locktype) >
```

11.6 lockinfo XML Element

Name: lockinfo
Namespace: http://www.iana.org/standards/dav/
Purpose: The lockinfo XML element is used with a LOCK method to specify the type of lock the client wishes to have created.

```
<!ELEMENT lockinfo (lockscope, locktype, owner?) >
```

11.7 lockscope XML Element

Name:	lockscope
Namespace:	http://www.iana.org/standards/dav/
Purpose:	Specifies whether a lock is an exclusive lock, or a shared lock.

```
<!ELEMENT lockscope (exclusive | shared) >
```

11.7.1 exclusive XML Element

Name:	exclusive
Namespace:	http://www.iana.org/standards/dav/
Purpose:	Specifies an exclusive lock

```
<!ELEMENT exclusive EMPTY >
```

11.7.2 shared XML Element

Name:	shared
Namespace:	http://www.iana.org/standards/dav/
Purpose:	Specifies a shared lock

```
<!ELEMENT shared EMPTY >
```

11.8 locktype XML Element

Name: locktype
Namespace: http://www.iana.org/standards/dav/
Purpose: Specifies the access type of a lock. At present, this specification only defines one lock type, the write lock.

```
<!ELEMENT locktype (write) >
```

11.8.1 write XML Element

Name: write
Namespace: http://www.iana.org/standards/dav/
Purpose: Specifies a write lock.

```
<!ELEMENT write EMPTY >
```

11.9 multistatus XML Element

Name: multistatus
Namespace: http://www.iana.org/standards/dav/
Purpose: Contains multiple response messages.
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.

```
<!ELEMENT multistatus (response+, responsedescription?) >
```

11.9.1 response XML Element

Name: response
Namespace: http://www.iana.org/standards/dav/
Purpose: Holds a single response describing the effect of a method on resource and/or its properties.
Description: A particular href MUST NOT appear more than once as the child of a response XML element under a multistatus XML element. This requirement is necessary in order to keep processing costs for a response to linear time. Essentially, this prevents having to search in order to group together all the responses by href. There are, however, no requirements regarding ordering based on href values.

```
<!ELEMENT response (href, ((href*, status)|(propstat+)),
responsedescription?) >
```

11.9.1.1 propstat XML Element

Name: propstat
Namespace: http://www.iana.org/standards/dav/
Purpose: Groups together a prop and status element that is associated with a particular href element.
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.

```
<!ELEMENT propstat (prop, status) >
```

11.9.1.2 status XML Element

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

```
<!ELEMENT status (#PCDATA) >
```

11.9.2 responsedescription XML Element

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

```
<!ELEMENT responsedescription (#PCDATA) >
```

11.10 owner XML Element

Name: owner
Namespace: http://www.iana.org/standards/dav/
Purpose: Provides information about the principal taking out a lock.
Description: 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.

```
<!ELEMENT owner (#PCDATA, ANY)* >
```

11.11 prop XML element

Name: prop
Namespace: http://www.iana.org/standards/dav/
Purpose: Contains properties related to a resource.
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.

```
<!ELEMENT prop ANY>
```

11.12 propertybehavior XML element

Name: propertybehavior
Namespace: http://www.iana.org/standards/dav/
Purpose: Specifies how properties are handled during a COPY or MOVE.
Description: The propertybehavior XML element specifies how properties are handled during a COPY or MOVE. If this XML element is not included in the request body then the server is expected to act as defined by the default property handling behavior of the associated method.

```
<!ELEMENT propertybehavior (omit | keepalive) >
```

11.12.1 keepalive XML element

Name: keepalive
Namespace: http://www.iana.org/standards/dav/
Purpose: Specifies requirements for the copying/moving of live properties.

Description: If a list of URIs is included as the value of keepalive then the named properties MUST be "live" after they are copied (moved) to the destination resource of a COPY (or MOVE). If the value "*" is given for the keepalive XML element, this designates that all live properties on the source resource MUST be live on the destination.
Value: `"*" ; #PCDATA value can only be "*"`

```
<!ELEMENT keepalive (#PCDATA | href+) >
```

11.12.2 omit XML element

Name: omit
Namespace: http://www.iana.org/standards/dav/
Purpose: Indicates that the associated method MAY succeed even if the server is not able to copy/move every property on the source resource, even in a dead form.
Description: The default behavior for a COPY or MOVE is to copy/move all properties or fail the method. In certain circumstances, such as when a server copies a resource over another protocol such as FTP, it may not be possible to copy/move the properties associated with the resource. Thus any attempt to copy/move over FTP would always have to fail because properties could not be moved over, even as dead properties. The omit XML element instructs the server that it should use best effort to copy properties but a failure to copy a property should not cause the method to fail.

```
<!ELEMENT omit EMPTY >
```

11.13 propertyupdate XML element

Name: propertyupdate
Namespace: http://www.iana.org/standards/dav/
Purpose: Contains a request to alter the properties on a resource.
Description: This XML element is a container for the information required to modify the properties on the resource. This XML element is multi-valued.

```
<!ELEMENT propertyupdate (remove | set)+ >
```

11.13.1 remove XML element

Name: remove
Namespace: http://www.iana.org/standards/dav/
Purpose: Lists the DAV properties to be removed from a resource.
Description: Remove instructs 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 XML elements in prop MUST be empty, as only the names of properties to be removed are required.

```
<!ELEMENT remove (prop) >
```

11.13.2 set XML element

Name: set
Namespace: http://www.iana.org/standards/dav/
Purpose: Lists the DAV property values to be set for a resource.
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.

```
<!ELEMENT set (prop) >
```

11.14 propfind XML Element

Name: propfind
Namespace: http://www.iana.org/standards/dav/
Purpose: Specifies the properties to be returned from a PROPFIND method. Two special elements are specified for use with propfind, allprop and propname.

```
<!ELEMENT propfind (allprop | propname | href+) >
```

11.14.1 allprop XML Element

Name: allprop
Namespace: http://www.iana.org/standards/dav/
Purpose: The allprop XML element specifies that all property names and values on the resource are to be returned.

```
<!ELEMENT allprop EMPTY >
```

11.14.2 propname XML Element

Name: propname
Namespace: http://www.iana.org/standards/dav/
Purpose: The propname XML element specifies that only a list of property names on the resource is to be returned.

```
<!ELEMENT propname EMPTY >
```

12 DAV Properties

For DAV properties, the name of the property is also the same as the name of the XML element which contains its value. In the section below, the final line of each section gives the element type declaration using the format defined in [Bray, Paoli, Sperberg-McQueen, 1998]. The "Value" field, where present, specifies futher restrictions on the allowable contents of the XML element using BNF (i.e., to further restrict the values of a PCDATA element).

12.1 creationdate Property

Name: creationdate
Namespace: http://www.iana.org/standards/dav/
Purpose: Records the time and date the resource was created.
Value: `;The time and date MUST be given in ISO 8601 format defined in Appendix 2`
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).

```
<!ELEMENT creationdate (#PCDATA) >
```

12.2 displayname Property

Name: displayname
Namespace: http://www.iana.org/standards/dav/
Purpose: Provides a name for the resource that is suitable for presentation to a user.

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.

```
<!ELEMENT displayname (#PCDATA) >
```

12.3 externalmembers Property

Name: externalmembers
Namespace: http://www.iana.org/standards/dav/
Purpose: Provides the list of external members defined on the resource.
Description: This property MUST be defined on any DAV compliant resource with external members. If defined it MUST contain the full list of external members. Resources MAY make this property read-only, thus only allowing its value to be altered using the ADDREF/DELREF methods.

```
<!ELEMENT externalmembers (href*) >
```

12.4 getcontentlanguage Property

Name: getcontentlanguage
Namespace: http://www.iana.org/standards/dav/
Purpose: Contains the Content-Language header returned by a GET without accept headers
Description: This property MUST be defined on any DAV compliant resource which supports GET, with the exception that if no Content-Language header is available, this property MUST NOT exist.
Value: `language-tag ;language-tag is defined in section 14.13 of [Fielding et al., 1997]`

```
<!ELEMENT getcontentlanguage (#PCDATA) >
```

12.5 getcontentlength Property

Name: getcontentlength
Namespace: http://www.iana.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.
Description: This property MUST be defined on any DAV compliant resource which returns the Content-Length header in response to a GET.
Value: `content-length ; see section 14.14 of [Fielding et al., 1997]`

```
<!ELEMENT getcontentlength (#PCDATA) >
```

12.6 getcontenttype Property

Name: getcontenttype
Namespace: http://www.iana.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.
Description: This property MUST be defined on any DAV compliant resource which returns the Content-Type header in response to a GET.
Value: `media-type   ; defined in Section 3.7 of [Fielding et al., 1997]`

```
<!ELEMENT getcontenttype (#PCDATA) >
```

12.7 getetag Property

Name: getetag
Namespace: http://www.iana.org/standards/dav/
Purpose: Contains the ETag header returned by a GET without accept headers.
Description: Note that the ETag on a 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 to a resource's access permissions may cause the ETag to change. This property MUST be defined on any DAV compliant resource which returns the Etag header in response to a GET, except for the case if no ETag header is returned, this property MUST NOT exist.
Value: `entity-tag  ; defined in Section 3.11 of [Fielding et al., 1997]`

```
<!ELEMENT getetag (#PCDATA) >
```

12.8 getlastmodified Property

Name: getlastmodified
Namespace: http://www.iana.org/standards/dav/
Purpose: Contains the Last-Modified header returned by a GET method without accept headers.
Description: Note that the last-modified date on a 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. This property MUST be defined on any DAV compliant resource which returns the Last-Modified header in response to a GET, except for the case if no Last-Modified header is returned, this property MUST NOT exist.
Value: `HTTP-date  ; defined in Section 3.3.1 of [Fielding et al., 1997]`

<!ELEMENT getlastmodified (#PCDATA) >

12.9 lockdiscovery Property

Name: lockdiscovery
Namespace: http://www.iana.org/standards/dav/
Purpose: Describes the active locks on a resource
Description: The lockdiscovery property returns a listing of who has a lock, what type of lock he has, 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.

```
<!ELEMENT lockdiscovery (activelock)* >
```

12.9.1 Example

>>Request

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

<?xml version="1.0"?>
<?namespace href="http://www.iana.org/standards/dav/" as="D"?>
<D:propfind>
      <D:prop><lockdiscovery/></D:prop>
</D:propfind>
```

>>Response

```
HTTP/1.1 207 Multi-Status
```

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

<?xml version="1.0"?>
<?namespace href ="http://www.iana.org/standards/dav/" as="D"?>
<D:multistatus>
       <D:response>
    <D:propstat>
              <D:prop>
                <D:lockdiscovery>
                  <D:activelock>
                    <D:locktype>write</D:locktype>
                      <D:lockscope>exclusive</D:lockscope>
                      <D:Depth>0</D:Depth>
                      <D:owner>Jane Smith</D:owner>
                      <D:timeout>Infinite</D:timeout>
                      <D:locktoken>
                            <D:href>
                opaquelocktoken:f81de2ad-7f3d-a1b2-4f3c-00a0c91a9d76
                     </D:href>
                      </D:locktoken>
                  </D:activelock>
                </D:lockdiscovery>
              </D:prop>
              <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
       </D:response>
</D:multistatus>
```

This resource has a single exclusive write lock on it, with an infinite timeout. Note that the Depth element could have been omitted as 0 is the default value of Depth.

12.10 resourcetype Property

Name: resourcetype
Namespace: http://www.iana.org/standards/dav/
Purpose: Specifies the nature of the resource.
Description: This property MUST be defined on all DAV compliant resources. The default value is empty.

```
<!ELEMENT resourcetype ANY >
```

12.11 source Property

Name: source
Namespace: http://www.iana.org/standards/dav/link/
Purpose: The destination of the source link identifies the resource that contains the unprocessed source of the link's source.
Description: 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.

```
<!ELEMENT source (link)* >
```

12.11.1 Example

```
<?xml version="1.0"?>
<?namespace href="http://www.iana.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, allowing element values to be enhanced without breaking older clients.

12.12 supportedlock Property

Name: supportedlock
Namespace: http://www.iana.org/standards/dav/
Purpose: To provide a listing of the lock capabilities supported by the resource.
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.

```
<!ELEMENT supportedlock (lockentry)* >
```

12.12.1 Example

>>Request

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


<?xml version="1.0"?>
<?namespace href="http://www.iana.org/standards/dav/" as="D"?>
<D:propfind>
       <D:prop><supportedlock/></D:prop>
</D:propfind>
```

>>Response

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

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

13 DAV Compliance Classes

A DAV compliant resource can choose from two classes of compliance. A client can discover the compliance classes of a resource 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 [Fielding et al., 1997].

Compliance classes are not necessarily sequential. A resource that is class 2 compliant MUST also be class 1 compliant; but if additional compliance classes are defined later, a resource that is class 1, 2, and 4 compliant might not be class 3 compliant.

13.1 Class 1

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

Class 1 compliant resources MUST return, at minimum, the value "1" in the DAV header on all responses to the OPTIONS method.

13.2 Class 2

A class 2 compliant resource MUST meet all class 1 requirements and support the supportedlock property as well as the LOCK method. It MUST also support the lockdiscovery property, since Section 12.9 specifies that the LOCK method MUST also support the lockdiscovery property.

Class 2 compliant resources MUST return, at minimum, the value "2" in the DAV header on all responses to the OPTIONS method.

14 Internationalization Considerations

In the realm of internationalization, this specification complies with the IETF Character Set Policy [Alvestrand, 1998]. In this specification, human-readable fields can be found either in 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 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 a language tagging capability for specifying the language of the contents of a particular XML element. XML uses either IANA registered language tags (see RFC 1766, [Alvestrand, 1995]) or ISO 639 language tags [ISO-639] in the "xml:lang" attribute of an XML element to identify the language its content and attributes.

Names used within this specification fall into three categories: names of 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., 425 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.

15 Security Considerations

This section is provided to detail issues concerning security implications of which WebDAV applications need to be aware.

All of the security considerations of HTTP/1.1 also apply to WebDAV. In addition, the security risks inherent in remote authoring require stronger authentication technology, and introduce several new privacy concerns, and may increase the hazards from poor server design. These issues are detailed below.

15.1 Authentication of Clients

Due to their emphasis on authoring, WebDAV servers need to use authentication technology to protect not just access to a network resource, but the integrity of the resource as well. Furthermore, the introduction of locking functionality requires support for authentication.

A password sent in the clear over an insecure channel is an inadequate means for protecting the accessibility and integrity of a resource as the password may be intercepted. Since Basic authentication for HTTP/1.1 performs essentially clear text transmission of a password, Basic authentication MUST NOT be used to authenticate a WebDAV client to a server unless the connection is secure. Furthermore, a WebDAV server MUST NOT send Basic authentication credentials in a WWW-Authenticate header unless the connection is secure. Examples of secure connections include a Transport Layer Security (TLS) connection, or a connection over a network which is physically secure, for example, an isolated network in a building with restricted access.

WebDAV applications MUST support the Digest authentication scheme [Franks, et al., 1997]. Since Digest authentication verifies that both parties to a communication know a shared secret, a password, without having to send that secret in the clear, Digest authentication avoids the security problems inherent in Basic authentication while providing a level of authentication which is useful in a wide range of scenarios.

15.2 Denial of Service

Denial of service attacks are of special concern to WebDAV servers. WebDAV plus HTTP enables denial of service attacks on every part of a system's resources.

The underlying storage can be attacked by PUTting extremely large files.

Asking for recursive operations on large collections can attack processing time.

Making multiple pipelined requests on multiple connections can attack network connections.

WebDAV servers need to be aware of the possibility of a denial of service attack at all levels.

15.3 Security through Obscurity

WebDAV provides, through the PROPFIND method, a mechanism for listing the member resources of a collection. This greatly diminishes the effectiveness of security or privacy techniques which rely only on the difficulty of discovering the names of network resources. Users of WebDAV servers are encouraged to use access control techniques to prevent unwanted access to resources, rather than depending on the relative obscurity of their resource names.

15.4 Privacy Issues Connected to Locks

When submitting a lock request a user agent may also submit an owner XML field giving contact information for the person taking out the lock (for those cases where a person, rather than a robot, is taking out the lock). This contact information is stored in a lockdiscovery property on the resource, and can be used by other collaborators to begin negotiation over access to the resource. However, in many cases this contact information can be very private, and should not be widely disseminated. Servers SHOULD limit read access to the lockdiscovery property as appropriate. Furthermore, user agents

SHOULD provide control over whether contact information is sent at all, and if contact information is sent, control over exactly what information is sent.

15.5 Privacy Issues Connected to Properties

Since property values are typically used to hold information such as the author of a document, there is the possibility that privacy concerns could arise stemming from widespread access to a resource's property data. To reduce the risk of inadvertent release of private information via properties, servers are encouraged to develop access control mechanisms that separate read access to the resource body and read access to the resource's properties. This allows a user to control the dissemination of their property data without overly restricting access to the resource's contents.

15.6 Reduction of Security due to Source Link

HTTP/1.1 warns against providing read access to script code because it may contain sensitive information. Yet WebDAV, via its source link facility, can potentially provide a URL for script resources so they may be authored. For HTTP/1.1, a server could reasonably prevent access to source resources due to the predominance of read-only access. WebDAV, with its emphasis on authoring, encourages read and write access to source resources, and provides the source link facility to identify the source. This reduces the security benefits of eliminating access to source resources. Users and administrators of WebDAV servers should be very cautious when allowing remote authoring of scripts, limiting read and write access to the source resources to authorized principals.

16 IANA Considerations

This document defines two namespaces, the namespace of property names, and the namespace of WebDAV-specific XML elements used within property values.

URLs are used for both names, for several reasons. Assignment of a URL does not require a request to a central naming authority, and hence allow WebDAV property names and XML elements to be quickly defined by any WebDAV user or application. URLs also provide a unique address space, ensuring that the distributed users of WebDAV will not have collisions among the property names and XML elements they create.

This specification defines a distinguished set of property names and XML elements which are understood by all WebDAV applications. The property names and XML elements in this specification are all derived from the base URL: http://www.iana.org/standards/dav/ by adding a suffix to this URL, for example, http://www.iana.org/standards/dav/creationdate for the "creationdate" property.

To ensure correct interoperation of this specification, IANA MUST reserve the URL namespace starting with http://www.iana.org/standards/dav/ for use by this specification, its revisions, and related WebDAV specifications.

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 [Bradner, 1996], Section 10.4, and describes the applicable copyright for this document.

19 Intellectual Property

The following notice is copied from RFC 2026 [Bradner, 1996], Section 10.4, and describes the position of the IETF concerning intellectual property claims made against this document.

of such proprietary rights by implementors or users of this specification can be obtained from the IETF Secretariat.

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

20 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, Brian Deen, 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.

Two from this list deserve 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. The contributions of Judith Slein in clarifying the requirements, and in patiently reviewing draft after draft, both improved this specification and expanded our minds on document management.

We would also like to thank John Turner for developing the XML DTD.

21 References

[Alvestrand, 1995] H. T. Alvestrand, "Tags for the Identification of Languages." RFC 1766. Uninett. March, 1995.

[Alvestrand, 1998] H. T. Alvestrand, "IETF Policy on Character Sets and Languages." RFC XXXX, BCP YY. Maxware. January, 1998.

[Bradner, 1996] S. Bradner, "The Internet Standards Process - Revision 3." RFC 2026, BCP 9. Harvard University. October, 1996.

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

[Bray, Paoli, Sperberg-McQueen, 1998] T. Bray, J. Paoli, C. M. Sperberg-McQueen, "Extensible Markup Language (XML)." World Wide Web Consortium Recommendation REC-XML-ZZZZ. http://www.w3.org/TR/PR-xml-971208.

[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.

[ISO-639] ISO (International Organization for Standardization). ISO 639:1988. "Code for the representation of names of languages."

[ISO-8601] ISO (International Organization for Standardization). ISO 8601:1988. "Data elements and interchange formats - Information interchange - Representation of dates and times."

[Lasher, Cohen, 1995] R. Lasher, D. Cohen, "A Format for Bibliographic Records," RFC 1807. Stanford, Myricom. June, 1995.

[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

[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, World Wide Web Consortium 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.

[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." RFC 2044. Alis Technologies. October, 1996.

22 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

23 Appendices

23.1 Appendix 1 - WebDAV Document Type Definition

This section provides a document type definition, following the rules in [Bray, Paoli, Sperberg-McQueen, 1998], for the XML elements used in the protocol stream and in the values of properties. It collects the element definitions given in Sections 11 and 12.

```
<!DOCTYPE webdav-1.0 [

<!--============ XML Elements from Section 11 ==================-->

<!ELEMENT activelock (locktype, lockscope, depth?, owner, timeout, locktoken)
>

<!ELEMENT lockentry (lockscope, locktype) >
<!ELEMENT lockinfo (lockscope, locktype, owner?) >

<!ELEMENT locktype (write) >
<!ELEMENT write EMPTY >

<!ELEMENT lockscope (exclusive | shared) >
<!ELEMENT exclusive EMPTY >
<!ELEMENT shared EMPTY >

<!ELEMENT depth (#PCDATA) >

<!ELEMENT owner (#PCDATA, ANY)* >

<!ELEMENT timeout (#PCDATA) >

<!ELEMENT locktoken (href) >

<!ELEMENT href (#PCDATA) >

<!ELEMENT link (src+, dst+) >
<!ELEMENT dst (#PCDATA) >
<!ELEMENT src (#PCDATA) >

<!ELEMENT multistatus (response+, responsedescription?) >

<!ELEMENT response (href, ((href*, status)|(propstat+)),
responsedescription?) >
<!ELEMENT status (#PCDATA) >
<!ELEMENT propstat (prop status) >
<!ELEMENT responsedescription (#PCDATA) >

<!ELEMENT prop ANY >

<!ELEMENT propertybehavior (omit | keepalive) >
<!ELEMENT omit EMPTY >
<!ELEMENT keepalive (#PCDATA | href+) >

<!ELEMENT propertyupdate (remove | set)+ >
<!ELEMENT remove (prop) >
<!ELEMENT set (prop) >

<!ELEMENT propfind (allprop | propname | href+) >
<!ELEMENT allprop EMPTY >
<!ELEMENT propname EMPTY >
<!ELEMENT collection EMPTY >


<!--=========== Property Elements from Section 12 ===============-->

<!ELEMENT creationdate (#PCDATA) >
<!ELEMENT displayname (#PCDATA) >
<!ELEMENT externalmembers (href*) >
<!ELEMENT getcontentlanguage (#PCDATA) >
<!ELEMENT getcontentlength (#PCDATA) >
<!ELEMENT getcontenttype (#PCDATA) >
<!ELEMENT getetag (#PCDATA) >
<!ELEMENT getlastmodified (#PCDATA) >
<!ELEMENT lockdiscovery (activelock)* >
<!ELEMENT resourcetype ANY >
<!ELEMENT source (link)* >
<!ELEMENT supportedlock (lockentry)* >

]>
```

23.2 Appendix 2 - ISO 8601 Date and Time Profile

The creationdate property specifies the use of the ISO 8601 date format. This section defines a profile of the ISO 8601 date format for use with this specification. This profile is quoted verbatim from draft-newman-datetime-01.txt (expired).

```
date-time       = full-date "T" full-time

full-date       = date-fullyear "-" date-month "-" date-mday
full-time       = partial-time time-offset

date-fullyear   = 4DIGIT
date-month      = 2DIGIT  ; 01-12
date-mday       = 2DIGIT  ; 01-28, 01-29, 01-30, 01-31 based on month/year
time-hour       = 2DIGIT  ; 00-23
time-minute     = 2DIGIT  ; 00-59
time-second     = 2DIGIT  ; 00-59, 00-60 based on leap second rules
time-secfrac    = "." 1*DIGIT
time-numoffset  = ("+" / "-") time-hour ":" time-minute
time-offset     = "Z" / time-numoffset

partial-time    = time-hour ":" time-minute ":" time-second
                 [time-secfrac]
```

Numeric offsets are calculated as local time minus UTC (Coordinated Universal Time). So the equivalent time in UTC can be determined by subtracting the offset from the local time. For example, 18:50:00-04:00 is the same time as 22:58:00Z.

If the time in UTC is known, but the offset to local time is unknown, this can be represented with an offset of "-00:00". This differs from an offset of "Z" which implies that UTC is the preferred reference point for the specified time.

23.3 Appendix 3 - Notes on Processing XML Elements

XML is a flexible data format that makes it easy to submit data that appears legal but in fact is not. The philosophy of "Be flexible in what you accept and strict in what you send" still applies, but it must not be applied inappropriately. XML is extremely flexible in dealing with issues of white space, element ordering, inserting new elements, etc. This flexibility does not require extension, especially not in the area of the meaning of elements.

There is no kindness in accepting illegal combinations of XML elements. At best it will cause an unwanted result and at worst it can cause real damage.

23.3.1 XML Syntax Error Example

The following request body for a PROPFIND method is illegal.

```
<?xml version="1.0"?>
<?namespace href ="http://www.iana.org/standards/dav/" as="D"?>
<D:propfind>
       <D:allprop/>
       <D:propname/>
</D:propfind>
```

The definition of the propfind element only allows for the allprop or the propname element, not both. Thus the above is an error and MUST be responded to with a 400 Bad Request.

Imagine, however, that a server wanted to be "kind" and decided to pick the allprop element as the true element and respond to it. A client running over a bandwidth limited line who intended to execute a propname would be in for a big surprise if the server treated the command as an allprop.

23.3.2 Unknown XML Element Example

The previous example was illegal because it contained two elements that were explicitly banned from appearing together in the propfind element. However, XML is an extensible language, so one can imagine new elements being defined for use with propfind. Below is the request body of a PROPFIND and, like the previous example, MUST be rejected with a 400 Bad Request by a server that does not understand the expired-props element.

```
<?xml version="1.0"?>
<?namespace href ="http://www.iana.org/standards/dav/" as="D"?>
<?namespace href="http://www.foo.bar/standards/props/" as="E"?>
<D:propfind>
       <E:expired-props/>
</D:propfind>
```

To understand why a 400 Bad Request is returned let us look at the request body as the server unfamiliar with expired-props sees it.

```
<?xml version="1.0"?>
<?namespace href ="http://www.iana.org/standards/dav/" as="D"?>
<?namespace href="http://www.foo.bar/standards/props/" as="E"?>
<D:propfind>
</D:propfind>
```

As the server does not understand the expired-props element, by the rules of XML, it MUST ignore it. Thus the server sees an empty propfind, which by the definition of the propfind element is illegal.

Please note that had the extension been additive it would not necessarily have resulted in a 400 Bad Request. For example, imagine the following request body for a PROPFIND:

```
<?xml version="1.0"?>
<?namespace href ="http://www.iana.org/standards/dav/" as="D"?>
<?namespace href="http://www.foo.bar/standards/props/" as="E"?>
<D:propfind>
       <D:propname/>
       <E:leave-out>*boss*</E:leave-out>
```

```
   </D:propfind>
```

The previous example contains the fictitious element leave-out. Its purpose is to prevent the return of any property whose name matches the submitted pattern. If the previous example were submitted to a server unfamiliar with leave-out, the only result would be that the leave-out element would be ignored and a propname would be executed.