WEBDAV Working Group
INTERNET DRAFT
<draft-ietf-webdav-protocol-07>

Expires August, 1998

Y.Y. Goland, Microsoft
E.J. Whitehead, Jr., UC Irvine
A. Faizi, Netscape
S.R. Carter, Novell
D. Jensen, Novell

March 6, 1998

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

Status of this Memo

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

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

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

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

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

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).

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 retrieve a hierarchical membership listing (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 instruct the server 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., 1998].

The sections below provide a detailed introduction to resource properties (section 3), collections of resources (section 4), and locking operations (section 5). These sections introduce the abstractions manipulated by the WebDAV-specific HTTP methods described in section 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.

The XML namespace extension (Appendix 4) is also used in this specification in order to allow for new XML elements to be added without fear of colliding with other element names.

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 response has been introduced to return status information for multiple resources. The 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 14), on internationalization support (section 15), and on security (section 16).

2 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].

3 Data Model for Resource Properties

3.1 The Resource Property Model

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

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

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

3.2 Existing Metadata Proposals

Properties have long played an essential role in the maintenance of large document repositories, and many current proposals contain some notion of a property, or discuss web metadata more generally. These include PICS [Miller et al., 1996], PICS-NG, XML, 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.

3.3 Properties and HTTP Headers

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

3.4 Property Values

The value of a property is, at minimum, well formed XML.

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.

3.5 Property Names

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

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

The XML namespace mechanism, which is based on URIs, is used to name properties because it 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.

3.6 Media Independent Links

Although HTML resources support links to other resources, the Web needs more general support for links between resources of any media type. WebDAV provides such links. A WebDAV link is a special type of property value, formally defined in section 11.4, that allows typed connections to be established between resources of any media type. The property value consists of source and destination Uniform Resource Locators (URLs); the property name identifies the link type.

4 Collections of Web Resources

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

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

4.1 Collection Resources

A collection is a resource whose state consists of an unordered list of internal members 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].

Any given internal 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.

WebDAV servers MUST treat HTTP URL namespaces as collections, regardless of whether they were created with the MKCOL method described in section 7.3.

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.

4.2 Creation and Retrieval of Collection Resources

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

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

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

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

4.3 HTTP URL Namespace Model

The HTTP URL Namespace is a hierarchical namespace where the hierarchy is delimited with the "/" character. DAV compliant resources MUST maintain the consistency of the HTTP URL namespace.

For example, if the collection http://www.foo.bar.org/a/ exists, but http://www.foo.bar.org/a/b/ does not exist, an attempt to create http://www.foo.bar.org/a/b/c must fail.

4.4 Source Resources and Output Resources

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

However, the server can sometimes process HTML resources before they are transmitted as a return entity body. For example, a server-side-include directive within an HTML file might 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 may be stored in a link on the output resource with type DAV:source (see section 12.10 for a description of the source link property). Storing the source URIs in links on the output resources places the burden of discovering the source on the authoring client. Note that the value of a source link is not guaranteed to point to the correct source. Source links may break or incorrect values may be entered. Also note that not all servers will allow the client to set the source link value. For example a server which generates source links on the fly for its CGI files will most likely not allow a client to set the source link value.

5 Locking

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

This specification allows locks to vary over two client-specified parameters, the number of principals involved (exclusive vs. shared) and the type of access to be granted. 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.

5.1 Exclusive Vs. Shared Locks

The most basic form of lock is an exclusive lock. This is a lock where the access right in question is only granted to a single principal. The need for this arbitration results from a desire to avoid having to 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 rights. 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 to the resource. Among those who have access permission to write to 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 locks are often too rigid. An exclusive lock is used to enforce a particular editing process: take out an exclusive 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.

5.2 Required Support

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

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

5.3 Lock Tokens

A lock token is a type of state token, represented as a URI, which 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 MUST 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.

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

5.4 opaquelocktoken Lock Token URI Scheme

The opaquelocktoken URI scheme is designed to be unique across all resources for all time. Due to this uniqueness quality, a client may submit an opaque lock token in an If 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 does not refer to an outstanding lock on the resource.

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 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]
```

5.5 Lock Capability Discovery

Since server lock support is optional, a client trying to lock a resource on a server can either try the lock and hope for the best, or perform some form of discovery to determine what lock capabilities the server supports. This is known as lock capability discovery. Lock capability discovery differs from discovery of supported access control types, since there may be access control types without corresponding lock types. A client can determine what lock types the server supports by retrieving the supportedlock property.

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

5.6 Active Lock Discovery

If another principal locks a resource that a principal wishes to access, it is useful for the second principal to be able to find out who the first principal is. For this purpose the lockdiscovery property is

provided. This property lists all outstanding locks, describes their type, and where available, provides their lock token.

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

5.7 Usage Considerations

Although the locking mechanisms specified here provide some help in preventing lost updates, they cannot guarantee that updates will never be lost. Consider the following scenario:

Two clients A and B are interested in editing the resource 'index.html'. Client A is an HTTP client rather than a WebDAV client, and so does not know how to perform locking.

Client A doesn't lock the document, but does a GET and begins editing.
Client B does LOCK, performs a GET and begins editing.
Client B finishes editing, performs a PUT, then an UNLOCK.
Client A performs a PUT, overwriting and losing all of B's changes.

There are several reasons why the WebDAV protocol itself cannot prevent this situation. First, it cannot force all clients to use locking because it must be compatible with HTTP clients that do not comprehend locking. Second, it cannot require servers to support locking because of the variety of repository implementations, some of which rely on reservations and merging rather than on locking. Finally, being stateless, it cannot enforce a sequence of operations like LOCK / GET / PUT / UNLOCK.

WebDAV servers that support locking can reduce the likelihood that clients will accidentally overwrite each other's changes by requiring clients to lock resources before modifying them. Such servers would effectively prevent HTTP 1.0 and HTTP 1.1 clients from modifying resources.

WebDAV clients can be good citizens by using a lock / retrieve / write /unlock sequence of operations (at least by default) whenever they interact with a WebDAV server that supports locking.

HTTP 1.1 clients can be good citizens, avoiding overwriting other clients' changes, by using entity tags in If-Match headers with any requests that would modify resources.

Information managers may attempt to prevent overwrites by implementing client-side procedures requiring locking before modifying WebDAV resources.

6 Write Lock

This section describes the semantics specific to the write lock type. The write lock is a specific instance of a lock type, and is the only lock type described in this specification.

6.1 Methods Restricted by Write Locks

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

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

6.2 Write Locks and Properties

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

6.3 Write Locks and Null Resources

It is possible to assert a write lock on a null resource in order to lock the name. A write locked null resource acts in all ways as a null resource, except that 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 MUST stay in the null state with the exception of the behavior described above.

If the resource is unlocked without a PUT, MKCOL, or similar method having been executed then the resource MUST return to its original NULL state.

A return to a full NULL state is generally interpreted as meaning that any attempt to execute a method on the resource will result in a 404 Not Found.

6.4 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 423 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 MUST be 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.

6.5 Write Locks and the If 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 a lock token MUST be submitted by an authorized principal in the If header for all locked resources that a method may interact with or the method MUST fail. For example, if a resource is to be moved and both the source and destination are locked then two lock tokens must be submitted, one for the source and the other for the destination.

6.5.1 Write Lock 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
If: <http://www.ics.uci.edu/users/f/fielding/index.html>
    (<opaquelocktoken:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>)
```

>>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 because the source resource is not modified by a COPY, and hence unaffected by the write lock. In this example, user agent authentication has previously occurred via a mechanism outside the scope of the HTTP protocol, in the underlying transport layer.

6.6 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 MUST NOT move the write lock with the resource although the resource is subject to being added to an existing lock as specified in section 6.4. For example, if the MOVE makes the resource a child of a collection that is depth locked then the resource will be under that collection's lock. Additionally, 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 will again be a member of the lock. In both these examples, as specified in section 6.5, an If header must be submitted containing a lock token for both the source and destination.

6.7 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 in the If header in order to make the request for a resource that is already locked.

However, a client may submit a LOCK method with an If header but without a body. This form of LOCK MUST only be used to "refresh" a lock. Meaning, at minimum, that any timers associated with the lock MUST be 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 SHOULD assume that the lock was not refreshed.

7 HTTP Methods for Distributed Authoring

The following new HTTP methods use XML as a request and response format. All DAV compliant clients and resources MUST use XML parsers that are compliant with [Bray, Paoli, Sperberg-McQueen, 1998]. All XML used in either requests or responses MUST be, at minimum, well formed. If a server receives ill-formed XML in a request it MUST reject the entire request with a 400 Bad Request. If a client receives ill-formed XML in a response then it MUST NOT assume anything about the outcome of the executed method and SHOULD treat the server as malfunctioning.

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 and the propfind XML element (section 11.14) along with all XML elements defined for use with that element.

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 PROPFIND request body MUST be treated as a request for the names and values of all properties.

All servers MUST support returning a response of content type text/xml that contains a multistatus XML element that describes the results of the attempts to retrieve the various properties.

If there is an error retrieving a property then a proper error result MUST be included in the response. A request to retrieve the value of a property which does not exist is an error and MUST be noted, if the response uses a multistatus XML element, 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 whether a particular property exists then the property should be silently excluded from the response.

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"?>
<?xml:namespace name="DAV:" as="D"?>
<?xml:namespace name="http://www.foo.bar/boxschema/" as="R"?>
<D:propfind>
       <D:prop>
              <R:bigbox/>
              <R:author/>
              <R:DingALing/>
              <R:Random/>
       </D:prop>
</D:propfind>
```

>>Response

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

<?xml version="1.0"?>
<?xml:namespace name="DAV:" as="D"?>
<?xml:namespace name="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"?>
<?xml:namespace name="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"?>
<?xml:namespace name="DAV:" as="D"?>
<?xml:namespace name="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:BoxType>Box type A</R:BoxType>
                        </R:bigbox>
                        <R:author>
                              <R:Name>Hadrian</R:Name>
                        </R:author>
                        <D:creationdate>
                              1997-12-01T17:42:21-08:00
                        </D:creationdate>
                        <D:displayname>
                              Example collection
                        </D:displayname>
                  <D:resourcetype><D:collection/></D:resourcetype>
                        <D:supportedlock>
                              <D:lockentry>
                                    <D:exclusive/><D:write/>
                              </D:lockentry>
                              <D:lockentry>
                                    <D:shared/><D:write/>
                              </D:lockentry>
                        </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>
                              <R:BoxType>Box type B</R:BoxType>
                        </R:bigbox>
                        <D:creationdate>
                              1997-12-01T18:27:21-08:00
                        </D:creationdate>
                        <D:displayname>
                              Example HTML resource
                        </D:displayname>
                        <D:getcontentlength>
                              4525
```

```
                              </D:getcontentlength>
                              <D:getcontenttype>
                                    text/html
                              </D:getcontenttype>
                              <D:getetag>
                                    zzyzx
                              </D:getetag>
                              <D:getlastmodified>
                                    Monday, 12-Jan-98 09:25:56 GMT
                              </D:getlastmodified>
                              <D:resourcetype/>
                              <D:supportedlock>
                                    <D:lockentry>
                                          <D:exclusive/><D:write/>
                                    </D:lockentry>
                                    <D:lockentry>
                                          <D:shared/><D:write/>
                                    </D:lockentry>
                              </D:supportedlock>
                        </D:prop>
                        <D:status>HTTP/1.1 200 OK</D:status>
                  </D:propstat>
            </D:response>
      </D: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 six properties defined on it:

http://www.foo.bar/boxschema/bigbox, http://www.foo.bar/boxschema/author, DAV:creationdate, DAV:displayname, DAV:resourcetype, and DAV:supportedlock.

The last four properties are WebDAV-specific, defined in section 12. Since GET is not supported on this resource, the get* properties (e.g., getcontentlength) 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 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), DAV:creationdate, DAV:displayname, DAV:getcontentlength, DAV:getcontenttype, DAV:getetag, DAV:getlastmodified, DAV:resourcetype, and 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 bytes (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"?>
<?xml:namespace name="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"?>
<?xml:namespace name="DAV:" as="D"?>
<?xml:namespace name="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: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:getcontentlength/>
                        <D:getcontenttype/>
                        <D:getetag/>
                        <D:getlastmodified/>
                        <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 six properties defined on it, http://www.foo.bar/boxschema/bigbox, http://www.foo.bar/boxschema/author, DAV:creationdate, DAV:displayname, DAV:resourcetype, and 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, DAV:creationdate, DAV:displayname, DAV:getcontentlength, DAV:getcontenttype, DAV:getetag, DAV:getlastmodified, DAV:resourcetype, and 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 the 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.

7.2.1 Status Codes for use with Multi-Status

The following are examples of response codes one would expect to be used in a Multi-Status response for this method. Note, however, that unless explicitly prohibited any 2/3/4/5xx series response code may be used in a Multi-Status response.

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.

423 Locked - The specified resource is locked and the client either is not a lock owner or the lock type requires a lock token to be submitted and the client did not submit it.

425 Insufficient Space on Resource - The server did not have sufficient space to record the 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"?>
<?xml:namespace name="DAV:" as="D"?>
<?xml:namespace name="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"?>
<?xml:namespace name="DAV:" as="D"?>
<?xml:namespace name="http://www.w3.com/standards/z39.50/" as="Z"?>
<D:multistatus>
       <D:response>
              <D:href>http://www.foo.com/bar.html</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, unless the Request-URI is "/". 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 SHOULD have 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 MUST NOT be cached as 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, or 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.

425 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 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 membership 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.

7.5 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.

7.6 DELETE

7.6.1 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.

7.6.2 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 with a DELETE on a collection with any value but infinity.

DELETE instructs that the collection specified in the request-URI 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.

If an error occurs with a resource other than the resource identified in the request URI then the response MUST be a 207 Multi-Status. 424 Method Failure errors SHOULD NOT be in the 207 Multi-Status. They can be safely left out because the client will know that the ancestors of a resource could not be deleted when the client receives an error for the ancestor's progeny. Additionally 204 No Content errors SHOULD NOT be returned in the 207 Multi-Status. The reason for this prohibition is that 204 No Content is the default success code.

7.6.2.1 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"?>
<?xml:namespace name="DAV:" as="d"?>
<d:multistatus>
       <d:response>
              <d:href>http://www.foo.bar/container/resource3</d:href>
              <d:status>HTTP/1.1 423 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. Thus the client knows that the attempt to delete http://ww.foo.bar/container/ must have also failed since the parent can not be deleted unless its child has also been deleted. Even though a Depth header has not been included, a depth of infinity is assumed because the method is on a collection.

7.7 PUT

7.7.1 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 affected. 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.

7.7.2 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.

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.

7.8 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.

All WebDAV compliant resources MUST support the COPY method. However, 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 be possible to copy a resource to a location that appears to be on the same server.

7.8.1 COPY for HTTP/1.1 resources

When the source resource is not a collection the result of the COPY method is the creation of a new resource at the destination whose state and behavior match that of the source resource as closely as possible. However, the exact state and behavior of the destination resource depend on what information the source resource is able to provide and what information the destination resource is able to accept.

Subsequent alterations to the destination resource will not modify the source resource. Subsequent alterations to the source resource will not modify the destination resource.

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.

7.8.2 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 subject to the effects of the propertybehavior XML element.

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.

7.8.3 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" Depth header behaviors.

A COPY of depth infinity instructs that the collection specified in the Request-URI 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 and its properties but not its internal members, are to be copied.

Any headers included with a COPY MUST be applied in processing every resource to be copied with the exception of the Destination header.

The Destination 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, the server MUST NOT copy any members of this collection. After detecting an 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.

If an error in executing the COPY method occurs with a resource other than the resource identified in the request URI then the response MUST be a 207 Multi-Status.

424 Method Failure errors SHOULD NOT be returned in the 207 Multi-Status from a COPY method. These responses can be safely omitted because the client will know that the progeny of a resource could not be copied when the client receives an error for the parent. Additionally 201 Created/204 No Content response codes SHOULD NOT be returned as values in 207 Multi-Status responses from COPY methods. They, too, can be safely omitted because they are the default success codes.

7.8.4 COPY and the Overwrite Header

If a resource exists at the destination and the Overwrite header is "T" then prior to performing the copy the server MUST perform a DELETE with Depth Infinity on the destination resource. If the Overwrite header is set to "F" then the operation will fail.

7.8.5 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.

412 Precondition Failed - The server was unable to maintain the liveness of the properties listed in the propertybehavior XML element or the Overwrite header is "F" and the state of the destination resource is non-null.

423 Locked - The destination resource was locked.

425 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.

502 Bad Gateway - This may occur when the destination is on another server and the destination server refuses to accept the resource.

7.8.6 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
```

7.8.7 No Overwrite Example

The following example shows the same copy operation being performed, but 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
```

7.8.8 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"?>
<?xml:namespace name="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"?>
<?xml:namespace name="DAV:" as="d"?>
<d:multistatus>
```

```
        <d:response>
              <d:href>http://www.foo.bar/othercontainer/R2/</d:href>
              <d:status>HTTP/1.1 412 Precondition Failed</d:status>
        </d:response>
  </d:multistatus>
```

The Depth header is unnecessary as the default behavior of COPY on a collection is to act as if a "Depth: infinity" header had been submitted. In this example most of the resources, along with the collection, were copied successfully. However the collection R2 failed, most likely due to a problem with maintaining the liveness of properties (this is specified by the propertybehavior XML element). Because there was an error copying R2, none of R2's members were copied. However no errors were listed for those members due to the error minimization rules given in section 7.8.3.

7.9 MOVE Method

The MOVE operation on a non-collection resource is the logical equivalent of a copy (COPY) followed by a delete of the source, where the actions are performed atomically. Consequently, the Destination header MUST be present on all MOVE methods and MUST follow all COPY requirements for the COPY part of the MOVE method. 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 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.

7.9.1 MOVE for Properties

The behavior of properties on a MOVE, including the effects of the propertybehavior XML element, MUST be the same as specified in section 7.8.2.

7.9.2 MOVE for Collections

A MOVE of depth infinity instructs that the collection specified in the Request-URI 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 MUST be applied in processing every resource to be moved with the exception of the Destination header.

The behavior of the Destination 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, the server MUST NOT move any members of the failed collection.. 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.

If an error occurs with a resource other than the resource identified in the request URI then the response MUST be a 207 Multi-Status.

424 Method Failure errors SHOULD NOT be returned as values in the 207 Multi-Status from a MOVE method. These errors can be safely omitted because the client will know that the progeny of a resource could not be moved when the client receives an error for the parent. Additionally 201 Created/204 No Content responses SHOULD NOT be returned as values in 207 Multi-Status responses from MOVES. These responses can be safely omitted because they are the default success codes.

7.9.3 MOVE and the Overwrite Header

If a resource exists at the destination and the Overwrite header is "T" then prior to performing the move the server MUST perform a DELETE with Depth infinity on the destination resource. If the Overwrite header is set to "F" then the operation will fail.

7.9.4 Status Codes

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

204 No Content - The source resource was successfully moved to a pre-existing destination resource.

412 Precondition Failed - The server was unable to maintain the liveness of the properties listed in the propertybehavior XML element or the Overwrite header is "F" and the state of the destination resource is non-null.

423 Locked - The source or the destination resource was locked.

502 Bad Gateway - This may occur when the destination is on another server and the destination server refuses to accept the resource.

7.9.5 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
```

7.9.6 Collection Example

>>Request

```
MOVE /container/ HTTP/1.1
Host: www.foo.bar
Destination: http://www.foo.bar/othercontainer/
Overwrite: F
If: (<opaquelocktoken:fe184f2e-6eec-41d0-c765-01adc56e6bb4>)
    (<opaquelocktoken:e454f3f3-acdc-452a-56c7-00a5c91e4b77>)
Content-Type: text/xml
Content-Length: xyz

<?xml version="1.0"?>
<?xml:namespace name="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"?>
<?xml:namespace name="DAV:" as="d"?>
<d:multistatus>
      <d:response>
            <d:href>http://www.foo.bar/othercontainer/C2/</d:href>
            <d:status>HTTP/1.1 423 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. Because there was an error copying /container/C2/, none of /container/C2's members were copied. However no errors were listed for those members due to the error minimization rules given in section 7.8.3. User agent authentication has previously occurred via a mechanism outside the scope of the HTTP protocol, in an underlying transport layer.

7.10 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.

Any resource which supports the LOCK method MUST, at minimum, support the XML request and response formats defined herein.

7.10.1 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.

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 contain the value of the lockdiscovery property in a prop XML element.

7.10.2 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 MUST also lock 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.

7.10.3 Locking Replicated Resources

Some servers automatically replicate resources across multiple URLs. In such a circumstance the server MUST only accept a lock on one of the URLs if the server can guarantee that the lock will be honored across all the URLs.

7.10.4 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 on a LOCK method. All resources that support the LOCK method MUST support 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 MUST return a single lock token which represents all the resources that have been locked. If an UNLOCK is successfully 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.

If no depth header is submitted on a LOCK request then the request MUST act as if a Depth of infinity had been submitted.

7.10.5 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.

7.10.6 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 refreshed.

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 refreshed.

7.10.7 Status Codes

200 Success - The lock request succeeded and the value of the lockdiscovery property is included in the body.

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.

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

7.10.8 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"?>
<?xml:namespace name="DAV:" as="D"?>
<D:lockinfo>
       <D:lockscope><D:exclusive/></D:lockscope>
       <D:locktype><D:write/></D:locktype>
       <D:owner>
              <D:href>http://www.ics.uci.edu/~ejw/contact.html</D:href>
       </D:owner>
</D:lockinfo>
```

>>Response

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

<?xml version="1.0"?>
<?xml:namespace name ="DAV:" as="D"?>
<D:prop>
       <D:lockdiscovery>
              <D:activelock>
                     <D:locktype><D:write/></D:locktype>
                     <D:lockscope><D:exclusive/></D:lockscope>
                     <D:depth>Infinity</D:depth>
                     <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). Note that the nonce, response, and opaque fields have not been calculated in the Authorization request header.

7.10.9 Example - Refreshing a Write Lock

>>Request

```
LOCK /workspace/webdav/proposal.doc HTTP/1.1
Host: webdav.sb.aol.com
Timeout: Infinite, Second-4100000000
If: (<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
Content-Type: text/xml
Content-Length: xxxxx

<?xml version="1.0"?>
<?xml:namespace name="DAV:" as="D"?>
<D:prop>
       <D:lockdiscovery>
              <D:activelock>
                     <D:locktype><D:write/></D:locktype>
```

```
                        <D:lockscope><D:exclusive/></D:lockscope>
                        <D:depth>Infinity</D:depth>
                        <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.

7.10.10 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"?>
<?xml:namespace name="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 Multi-Status
Content-Type: text/xml
Content-Length: xxxxx

<?xml version="1.0"?>
<?xml:namespace name="DAV:" as="D"?>
<D:multistatus>
       <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 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.

7.11 UNLOCK Method

The UNLOCK method removes the lock identified by the lock token in the Lock-Token request header from the Request-URI, and all other resources included in the lock. If all resources which have been locked under the submitted lock token can not be unlocked then the UNLOCK request MUST fail.

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

7.11.1 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 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 14 for more details. In general, however, support for one compliance class does not entail support for any other.

8.2 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 unless the particular method explicitly provides such guarantees.

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 that has a defined interaction with the Depth header MUST be applied to all resources in the scope of the method except where alternative behavior is explicitly defined. 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 with a depth header 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 If 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 MUST be 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, will 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.

8.3 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.

8.4 If Header

```
If = "If" ":" ( 1*No-tag-list | 1*Tagged-list)
No-tag-list = List
Tagged-list = Resource 1*List
Resource = Coded-url
List = "(" 1*(["Not"](State-token | "[" entity-tag "]")) ")"
State-token = Coded-url
Coded-url = "<" URI ">"
```

The If 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 header is intended for use with any URI which represents state information, referred to as a state token, about a resource as well as e-tags. A typical example of a state token is a lock token, and lock tokens are the only state tokens defined in this specification.

All DAV compliant resources MUST honor the If header.

The If header's purpose is to describe a series of state lists. If the state of the resource to which the header is applied does not match any of the specified state lists then the request MUST fail with a 412 Precondition Failed. If one of the described state lists matches the state of the resource then the request may succeed.

8.4.1 No-tag-list Production

The No-tag-list production describes a series of state tokens and e-tags. If multiple No-tag-list productions are used then only one needs to match the state of the resource for the method to be allowed to continue.

If a method, due to the presence of a Depth or Destination header, is applied to multiple resources then the No-tag-list production MUST be applied to each resource the method is applied to.

For example:

```
If: (<locktoken:a-write-lock-token> ["I am an e-tag"]) (["I am another e-
tag"])
```

The previous header would require that any resources within the scope of the method must either be locked with the specified lock token and in the state identified by the "I am an e-tag" e-tag or in the state identified by the second e-tag "I am another e-tag". To put the matter more plainly one can think of the previous If header as being in the form (or (and <locktoken:a-write-lock-token> ["I am an e-tag"]) (and ["I am another e-tag"])).

8.4.2 Tagged-list Production

The tagged-list production scopes a list production. That is, it specifies that the lists following the resource specification only apply to the specified resource. The scope of the resource production begins with the list production immediately following the resource production and ends with the next resource production, if any.

When the If header is applied to a particular resource, the Tagged-list productions MUST be searched to determine if any of the listed resources match the operand resource(s) for the current method. If none of the resource productions match the current resource then the header MUST be ignored. If one of the resource productions does match the name of the resource under consideration then the list productions following the resource production MUST be applied to the resource in the manner specified in the previous section.

The same URI MUST NOT appear more than once in a resource production in an If header.

For example:

```
COPY /resource1 HTTP/1.1
Host: www.foo.bar
Destination: http://www.foo.bar/resource2
If: <http://www.foo.bar/resource1> (<locktoken:a-write-lock-token> [W/"A
weak e-tag"]) (["strong e-tag"]) <http://www.bar.bar/random>(["another
strong e-tag"])
```

In this example http://www.foo.bar/resource1 is being copied to http://www.foo.bar/resource2. When the method is first applied to http://www.foo.bar/resource1, resource1 must be in the state specified by "(<locktoken:a-write-lock-token> [W/"A weak e-tag"]) (["strong e-tag"])", that is, it either must be locked with a lock token of "locktoken:a-write-lock-token" and have a weak entity tag W/"A weak e-tag" or it must have a strong entity tag "strong e-tag".

That is the only success condition since the resource http://www.bar.bar/random never has the method applied to it (the only other resource listed in the If header) and http://www.foo.bar/resource2 is not listed in the If header.

8.4.3 not Production

Every state token or e-tag is either current, and hence describes the state of a resource, or is not current, and does not describe the state of a resource. The boolean operation of matching a state token or e-tag to the current state of a resource thus resolves to a true or false value. The not production is used to reverse that value. The scope of the not production is the state-token or entity-tag immediately following it.

```
If: (Not <locktoken:write1> <locktoken:write2>)
```

When submitted with a request, this If header requires that all operand resources must not be locked with locktoken:write1 and must be locked with locktoken:write2.

8.4.4 Matching Function

When performing If header processing, the definition of a matching state token or entity tag is as follows.

Matching entity tag: Where the entity tag matches an entity tag associated with that resource.

Matching state token: Where there is an exact match between the state token in the If header and any state token on the resource.

8.4.5 If Header and Non-DAV Compliant Proxies

Non-DAV compliant proxies will not honor the If header, since they will not understand the If header, and HTTP requires non-understood headers to be ignored. When communicating with HTTP/1.1 proxies, the "Cache-Control: no-cache" request header MUST be used so as to prevent the proxy from improperly trying to service the request from its cache. When dealing with HTTP/1.0 proxies the "Pragma: no-cache" request header MUST be used for the same reason.

8.5 Lock-Token Request Header

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

The Lock-Token request header is used with the UNLOCK method to identify the lock to be removed. The lock token in the Lock-Token request header MUST identify a lock that contains the resource identified by Request-URI as a member.

8.6 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. If the overwrite header is not included in a COPY or MOVE request then the resource MUST treat the request as if it has an overwrite header of value "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.

All DAV compliant resources MUST support the Overwrite header.

8.7 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.

8.8 Timeout Request 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.

Timeout response valuse 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 will elapse between granting of the lock at the server, and the automatic removal of the lock. The timeout value for timetype "Second" MUST NOT be 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 may be lost. Specifically, if the server wishes to harvest the lock upon time-out, 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.

A client MUST NOT assume that just because the time-out has expired the lock has been lost.

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 Locked

The source or destination resource of a method is locked.

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 Insufficient Space on Resource

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

10 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 which contain 200, 300, 400, and 500 series status codes generated during the method invocation. 100 series status codes SHOULD 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: DAV:
Purpose: Describes a lock on a resource.

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

11.1.1 depth XML Element

Name: depth
Namespace: DAV:
Purpose: The value of the depth header used to create a lock.
Value: `"0" | "infinity"`

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

11.1.2 locktoken XML Element

Name: locktoken
Namespace: DAV:
Purpose: The lock token associated with a lock.
Description: The href contains one or more opaque lock token URIs which all refer to the same lock (i.e., the OpaqueLockToken-URI production in section 5.4).

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

11.1.3 timeout XML Element

Name: timeout
Namespace: DAV:
Purpose: The timeout associated with a lock
Value: `TimeType ;Defined in section 8.8`

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

11.2 collection XML Element

Name: collection
Namespace: 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: 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: 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. The values in the href XML elements inside the src and dst XML elements of the link XML element MUST NOT be rejected if they point to resources which do not exist.

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

11.4.1 dst XML Element

Name: dst
Namespace: DAV:
Purpose: Indicates the destination of a link
Value: `URI`

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

11.4.2 src XML Element

Name: src
Namespace: DAV:
Purpose: Indicates the source of a link.
Value: URI

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

11.5 lockentry XML Element

Name: lockentry
Namespace: 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: 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: 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: DAV:
Purpose: Specifies an exclusive lock

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

11.7.2 shared XML Element

Name: shared
Namespace: DAV:
Purpose: Specifies a shared lock

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

11.8 locktype XML Element

Name: locktype
Namespace: 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: DAV:
Purpose: Specifies a write lock.

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

11.9 multistatus XML Element

Name: multistatus
Namespace: 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: 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: DAV:
Purpose: Groups together a prop and status element that is associated with a particular href element.
Description: The propstat XML element MUST contain one or more empty prop 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: 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: 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: 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 ANY>
```

11.11 prop XML element

Name: prop
Namespace: 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 a prop XML element 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: 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. All WebDAV compliant resources MUST support the propertybehavior XML element.

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

11.12.1 keepalive XML element

Name: keepalive
Namespace: 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. If the requirements specified by the keepalive element can not be honored then the method MUST fail with a 412 Precondition Failed. All DAV compliant resources MUST support the keepalive XML element for use with the COPY and MOVE methods.
Value: `"*" ; #PCDATA value can only be "*"`

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

11.12.2 omit XML element

Name: omit
Namespace: DAV:
Purpose: The omit XML element instructs the server that it should use best effort to copy properties but a failure to copy a property MUST NOT cause the method to fail.
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. All DAV compliant resources MUST support the omit XML element on COPY/MOVE methods.

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

11.13 propertyupdate XML element

Name: propertyupdate
Namespace: 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: 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 a prop XML element inside of a remove XML element 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: DAV:
Purpose: Lists the DAV property values to be set for a resource.
Description: The set XML element MUST contain only a prop XML element. The elements contained by the prop XML element inside the set XML element MUST 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: DAV:
Purpose: Specifies the properties to be returned from a PROPFIND method. Two special elements are specified for use with propfind, allprop and propname. If prop is used inside propfind it MUST only contain property names, not values.

```
<!ELEMENT propfind (allprop | propname | prop) >
```

11.14.1 allprop XML Element

Name: allprop
Namespace: 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: 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 that 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: DAV:
Purpose: Records the time and date the resource was created.
Value: `date-time ; See Appendix 2`
Description: The creationdate 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: DAV:
Purpose: Provides a name for the resource that is suitable for presentation to a user.
Description: The displayname 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 getcontentlanguage Property

Name: getcontentlanguage
Namespace: DAV:
Purpose: Contains the Content-Language header returned by a GET without accept headers
Description: The getcontentlanguage property MUST be defined on any DAV compliant resource that returns the Content-Language header on a GET.
Value: `language-tag ;language-tag is defined in section 14.13 of [Fielding et al., 1997]`

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

12.4 getcontentlength Property

Name: getcontentlength
Namespace: DAV:
Purpose: Contains the Content-Length header returned by a GET without accept headers.
Description: The getcontentlength property MUST be defined on any DAV compliant resource that 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) >
```

12.5 getcontenttype Property

Name: getcontenttype
Namespace: DAV:
Purpose: Contains the Content-Type header returned by a GET without accept headers.
Description: This getcontenttype property MUST be defined on any DAV compliant resource that 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) >
```

12.6 getetag Property

Name: getetag
Namespace: 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. The getetag property MUST be defined on any DAV compliant resource that returns the Etag header in response to a GET.
Value: `entity-tag ; defined in section 3.11 of [Fielding et al., 1997]`

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

12.7 getlastmodified Property

Name: getlastmodified
Namespace: 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. The getlastmodified property MUST be defined of any DAV compliant resource that returns the Last-Modified header in response to a GET.
Value: `HTTP-date ; defined in section 3.3.1 of [Fielding et al., 1997]`

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

12.8 lockdiscovery Property

Name: lockdiscovery
Namespace: 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.

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

12.8.1 Example

>>Request

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

<?xml version="1.0"?>
<?xml:namespace name="DAV:" as="D"?>
<D:propfind>
       <D:prop><D:lockdiscovery/></D:prop>
</D:propfind>
```

>>Response

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

<?xml version="1.0"?>
<?xml:namespace name="DAV:" as="D"?>
<D:multistatus>
       <D:response>
              <D:href>http://www.foo.bar/container/</D:href>
              <D:propstat>
                     <D:prop>
                            <D:lockdiscovery>
                                   <D:activelock>
                                   <D:locktype><D:write/></D:locktype>
                                   <D:lockscope><D: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.

12.9 resourcetype Property

Name: resourcetype
Namespace: DAV:
Purpose: Specifies the nature of the resource.
Description: The resourcetype property MUST be defined on all DAV compliant resources. The default value is empty.

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

12.10 source Property

Name: source
Namespace: DAV:
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)* >
```

12.10.1 Example

```
<?xml version="1.0"?>
<?xml:namespace name="DAV:" as="D"?>
<?xml:namespace name="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.

12.11 supportedlock Property

Name: supportedlock
Namespace: 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.

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

12.11.1 Example

>>Request

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

<?xml version="1.0"?>
<?xml:namespace name="DAV:" as="D"?>
<D:propfind>
       <D:prop><D:supportedlock/></D:prop>
</D:propfind>
```

>>Response

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

<?xml version="1.0"?>
<?xml:namespace name="DAV:" as="D"?>
<D:multistatus>
       <D:response>
              <D:href>http://www.foo.bar/container/</D:href>
              <D:propstat>
                     <D:prop>
                            <D:supportedlock>
                                   <D:lockentry>
                                   <D:lockscope><D:exclusive/></D:lockscope>
                                   <D:locktype><D:write/></D:locktype>
                                   </D:lockentry>
                                   <D:lockentry>
                                   <D:lockscope><D:shared/></D:lockscope>
                                   <D:locktype><D:write/></D:locktype>
                                   </D:lockentry>
                            </D:supportedlock>
                     </D:prop>
                     <D:status>HTTP/1.1 200 OK</D:status>
              </D:propstat>
       </D:response>
</D:multistatus>
```

13 DAV XML Processing Instructions

All DAV compliant resources MUST ignore any unknown XML element and all its children encountered while processing a DAV method that uses XML as its command language.

This restriction also applies to the processing, by clients, of DAV property values where unknown XML elements SHOULD be ignored unless the property's schema declares otherwise.

This restriction does not apply to setting dead DAV properties on the server where the server MUST record unknown XML elements.

Additionally, this restriction does not apply to the use of XML where XML happens to be the content type of the entity body, for example, when used as the body of a PUT.

14 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. Also note that identifiers other than numbers may be used as compliance class identifiers.

14.1 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.

14.2 Class 2

A class 2 compliant resource MUST meet all class 1 requirements and support the LOCK method, the supportedlock property, the lockdiscovery property, the Time-Out response header and the Lock-Token request header. A class "2" compliant resource SHOULD also support the Time-Out request header and the owner XML element.

Class 2 compliant resources MUST return, at minimum, the values "1" and "2" in the DAV header on all responses to the OPTIONS method.

15 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, at minimum, using the UTF-8 [Yergeau, 1998] encoding of the ISO 10646 multilingual plane.

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 of its content and attributes.

WebDAV applications MUST support the character set tagging, character set encoding, and the language tagging functionality of the XML specification.

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., 423 Locked). While the possibility exists that a poorly crafted user agent would display this message to a user, internationalized applications will ignore this message, and display an appropriate message in the user's language and character set.

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

16 Security Considerations

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, introduce several new privacy concerns, and may increase the hazards from poor server design. These issues are detailed below.

16.1 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.

16.2 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.

16.3 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 that 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.

16.4 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.

16.5 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.

16.6 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.

17 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 that are understood by all WebDAV applications. The property names and XML elements in this specification are all derived from the base URI DAV: by adding a suffix to this URI, for example, DAV:creationdate for the "creationdate" property.

This specification also defines a URI scheme for the encoding of lock tokens, the opaquelocktoken URI scheme described in section 5.4.

To ensure correct interoperation based on this specification, IANA must reserve the URI namespaces starting with "DAV:" and with "opaquelocktoken:" for use by this specification, its revisions, and related WebDAV specifications.

18 Terminology

Collection - A resource that contains member resources and meets the requirements in section 4 of this specification.

Member Resource - A resource contained by a collection.

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

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.

19 Copyright

The following copyright notice is copied from RFC 2026 [Bradner, 1996], section 10.4, and describes the applicable copyright for this document.

Copyright (C) The Internet Society March 6, 1998. All Rights Reserved.

This document and translations of it may be copied and furnished to others, and derivative works that comment on or otherwise explain it or assist in its implementation may be prepared, copied, published and distributed, in whole or in part, without restriction of any kind, provided that the above copyright notice and this paragraph are included on all such copies and derivative works. However, this document itself may not be modified in any way, such as by removing the copyright notice or references to the Internet Society or other Internet organizations, except as needed for the purpose of developing Internet standards in which case the procedures for copyrights defined in the Internet Standards process must be followed, or as required to translate it into languages other than English.

The limited permissions granted above are perpetual and will not be revoked by the Internet Society or its successors or assignees.

This document and the information contained herein is provided on an "AS IS" basis and THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

20 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.

The IETF takes no position regarding the validity or scope of any intellectual property or other rights that might be claimed to pertain to the implementation or use other technology described in this document or the extent to which any license under such rights might or might not be available; neither does it represent that it has made any effort to identify any such rights. Information on the IETF's procedures with respect to rights in standards-track and standards-related documentation can be found in BCP-11. Copies of claims of rights made available for publication and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by 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.

21 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, Sanford Barr, 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.

22 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 2277, BCP 18. Uninett. 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, Hollander, Layman, 1998] T. Bray, D. Hollander, A. Layman, "Name Spaces in XML" World Wide Web Consortium Note, http://www.w3.org/TR/1998/NOTE-xml-names.

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

[Franks et al., 1997] J. Franks, P. Hallam-Baker, J. Hostetler, P. Leach, A. Luotonen, E. Sink, and L. Stewart. "An Extension to HTTP : Digest Access Authentication" RFC 2069. Northwestern University, CERN, Spyglass Inc., Microsoft Corp., Netscape Communications Corp., Spyglass Inc., Open Market Inc. January 1997.

[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, 1998] P. J. Leach, R. Salz, "UUIDs and GUIDs." Internet-draft, work-in-progress, February, 1998. ftp://ietf.org/internet-drafts/draft-leach-uuids-guids-01.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., 1998] J. A. Slein, F. Vitali, E. J. Whitehead, Jr., D. Durand, "Requirements for Distributed Authoring and Versioning Protocol for the World Wide Web." RFC 2291. Xerox, Univ. of Bologna, U.C. Irvine, Boston Univ. February, 1998.

[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, 1998] F. Yergeau, "UTF-8, a transformation format of Unicode and ISO 10646." RFC 2279. Alis Technologies. January, 1998.

23 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

24 Appendices

24.1 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 (lockscope, locktype, 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 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 | prop) >
<!ELEMENT allprop EMPTY >
<!ELEMENT propname EMPTY >
```

```
<!ELEMENT collection EMPTY >


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

<!ELEMENT creationdate (#PCDATA) >
<!ELEMENT displayname (#PCDATA) >
<!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)* >

]>
```

24.2 Appendix 2 - ISO 8601 Date and Time Profile

The creationdate property specifies the use of the ISO 8601 date format [ISO-8601]. 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.

24.3 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.

24.3.1 XML Syntax Error Example

The following request body for a PROPFIND method is illegal.

```
<?xml version="1.0"?>
<?xml:namespace name="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.

Additionally, if a server were lenient and decided to reply to this request, the results would vary randomly from server to server, with some servers executing the allprop directive, and others executing the propname directive. This reduces interoperability rather than increasing it.

24.3.2 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"?>
<?xml:namespace name="DAV:" as="D"?>
<?xml:namespace name="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"?>
<?xml:namespace name="DAV:" as="D"?>
<?xml:namespace name="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"?>
<?xml:namespace name="DAV:" as="D"?>
<?xml:namespace name="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.

24.4 Appendix 4 -- XML Namespaces for WebDAV

[NOTE TO RFC EDITOR: If, as expected, the World Wide Web Consortium issues XML namespaces as a W3C Recommendation before this document is published as an RFC (i.e., after approval by the IESG, but before appearing in the rfc directory), then the text of this appendix must be changed to read:

XML namespace functionality in this specification MUST conform to W3C Recommendation, "Name Spaces in XML" REC-XML-NAMES-1998????.
]

24.4.1 Introduction

To provide a unique space of XML element names which has decentralized extensibility, this specification uses a feature of XML known as XML "namespaces". This appendix provides a normative reference for XML namespace functionality for implementations of this specification. All DAV compliant systems MUST support the XML namespace extension as specified in this appendix."

The remainder of this appendix is intended to match, as closely as needed, the text in Note-xml-names-19980119, "Name Spaces in XML", edited by Tim Bray, Dave Hollander, and Andrew Layman, http://www.w3.org/TR/1998/NOTE-xml-names. To meet this goal, the text in this appendix is mostly quoted verbatim from this source. As future drafts of the XML namespace proposal are generated, this appendix will be updated. To ensure this appendix reflects the exact XML namespace proposal, the notational conventions and BNF productions in this appendix match those of the XML specification [Bray, Paoli, Sperberg-McQueen, 1998].

XML Namespaces are based on the use of qualified names. Names are permitted to contain a colon, separating the name into two parts, the namespace name and the local name. The namespace name identifies a schema's URI. The combination of the universally-managed URI namespace and the local schema namespace produces names that are guaranteed universally unique.

XML syntax does not allow direct use of a URI as a namespace name, because URIs can contain characters not allowed in XML element names. Consequently, the namespace name serves as a proxy for a URI. A special processing instruction described below is used to declare the association of the namespace name with a URI; software that supports this namespace proposal MUST recognize and act on namespace processing instructions.

A namespace is declared using a reserved processing instruction.

24.4.2 Namespace Declaration PI

```
[1]  NamespacePI ::=  '<?xml:namespace' S 'name=' SystemLiteral S 'href='
SystemLiteral S 'as=' NSName S? '?>'
[2]  NSName ::=  ' Name ' | " Name "
```

The "name" SystemLiteral is a URI which uniquely identifies the namespace. The "href" SystemLiteral is an optional URI which may be used to retrieve the schema, if one is provided. Some namespaces need no schemas; this specification does not depend on their existence, or on the use of any particular machine- or human-readable syntax in the schema.

The NSName gives the namespace name which will be used as a link to associate names in an XML document with this schema.

To accomplish this, the production for prolog is replaced as follows:

24.4.3 Prolog with Namespace Declarations

```
[3]  prolog ::=  XMLDecl? S? NamespacePI* Misc* (doctypedecl Misc*)? [  wfc:
Unique Namespace Names ]
```

24.4.4 Well-Formedness Constraint - Unique Namespace Names

No namespace name may be declared more than once.

24.4.5 Qualified Names

Within the document, some names (constructs corresponding to the nonterminal Name) are replaced by qualified names, defined as follows:

```
[4]  QName ::=  (NSPart ':')? LocalPart
[5]  NSPart ::=  Name [  wfc: Namespace Name Declared ]
[6]  LocalPart ::=  Name
```

The NSPart provides the namespace name part of the qualified name, and may be associated with defining schema through the URI in the applicable namespace declaration.

The LocalPart provides the local name part of the qualified name.

24.4.6 Well-Formedness Constraint - Namespace Name Declared

The namespace name, unless it is "xml", must have been declared in a namespace declaration. The namespace name xml is reserved, and considered to have been implicitly declared.

24.4.7 Using Qualified Names

To enable the proper use of qualified names, it is necessary to banish colons from all Names which are not qualified; two productions are replaced as follows:

```
[7]  Name ::=  (Letter | '_' ) (NameChar)*
[8]  MiscName ::=  '.' | '-' | '_' | CombiningChar | Ignorable | Extender
```

24.4.8 Element Names

Element types may be given as qualified names. To do this, the productions for start-, end-, and empty-element tags (STag, ETag, and EmptyElement) are replaced as follows:

```
[9]  STag ::=  '<' QName (S Attribute)* S? '>'
[10]  ETag ::=  '</' QName S? '>'
[11]  EmptyElement ::=  '<' QName (S Attribute)* S? '/>'
```

24.4.9 Scope and Meaning of Qualified Names

[Note to the reader: This section does not appear in NOTE-xml-names, but is necessary to avoid ambiguity for WebDAV XML processors.]

WebDAV compliant XML processors MUST interpret a qualified name as a URI constructed by appending the LocalPart to the schema URI of the namespace. The scope of a namespace in a qualified name is limited to a single element tag. Every start tag, end tag, or empty XML element from a namespace MUST include the namespace name in the tag.

Scope Example

```
<?xml:namespace name="http://www.del.jensen.org/" as="del"?>
<del:glider>
       <del:glidername>
              Johnny Updraft
       </del:glidername>
       <del:glideraccidents/>
</del:glider>
```

In this example, the qualified element name "del:glider" is interpreted as the URL "http://www.del.jensen.org/glider". Since the scope of a namespace is limited to a single element, each start tag, end tag, and empty element tag in the example includes the short name of the namespace, "del" as part of the qualified name.

```
<?xml:namespace name="http://www.del.jensen.org/" as="bar"?>
<bar:glider>
       <bar:glidername>
              Johnny Updraft
       </bar:glidername>
       <bar:glideraccidents/>
</bar:glider>
```

Even though this example is syntactically different from the previous example, it is semantically identical. Each instance of the namespace name "bar" is replaced with "http://www.del.jensen.org/" and then appended to the local name for each element tag. The resulting tag names in this example are exactly the same as for the previous example.

```
<?xml:namespace name="http://www.del.jensen.org/glide" as="foo"?>
<foo:r>
       <foo:rname>
              Johnny Updraft
       </foo:rname>
       <foo:raccidents/>
</foo:r>
```

This example is semantically identical to the two previous ones. Each instance of the namespace name "foo" is replaced with "http://www.del.jensen.org/glide" which is then appended to the local name for each element tag, the resulting tag names are identical to those in the previous examples.