WEBDAV Working Group Y. Y. Goland, Microsoft
INTERNET-DRAFT E. J. Whitehead, Jr., U.C. Irvine
<draft-ietf-webdav-protocol-04> A. Faizi, Netscape
S. R Carter, Novell
D. Jensen, Novell
Expires April 20, 1998 October 12, 1997

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

Status of this Memo

Abstract

This Document specifies a set of methods and content-types ancillary to HTTP/1.1 for the management of resource properties, simple name space manipulation, simple resource locking (collision avoidance) and resource version control.

Table of Contents

1 Terminology

Collection - A resource that contains member resources.

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

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

External Member Resource - a member resource with an absolute URI that is not relative to its parent s URI.

Properties - A set of name/value pairs that contain descriptive information about a resource.

Live Properties - Properties whose semantics and syntax are enforced by the server. For example, a live "read-only" property that is enforced by the server would disallow PUTs to the associated resource.

Dead properties - Properties whose semantics and syntax are not enforced by the server. A dead "read-only" property would not be enforced by the server and thus would not be used by the server as a reason to disallow a PUT on the associated resource.

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].

2 Data Model and Methods for DAV Properties

2.1 Introduction

2.1.1 The DAV Property

Properties are pieces of data that describe the state of a resource. Properties are data about data. The term property is used within this specification to disambiguate the concept from the overloaded terms "metadata" and "attribute".

Properties are used within 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.

2.1.2 Existing Metadata Proposals

Properties have a long played an essential role in the maintenance of large document repositories, and many current proposals contain some notion of a property. These include PICS [Miller et al., 1996], PICS-NG, the Rel/Rev draft [Maloney, 1996], Web Collections, XML [Bray, Sperberg-McQueen, 1997], several proposals on representing relationships within HTML,

digital signature manifests (DCMF), and a position paper on Web metadata architecture [Berners-Lee, 1997].

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, RFC 1807 [Lasher, Cohen, 1995], a technical report bibliographic format employed by the Dienst system, and the proceedings from the first IEEE Metadata conference describe many community-specific metadata sets.

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

2.1.3 Properties and HTTP Headers

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

2.2 A Property Model for HTTP Resources

2.2.1 Overview

The DAV property model is based on name/value doubles. The name of a property identifies the property's syntax and semantics, and provides an address with which to refer to a property. The name and value of a property is expressed as a well-formed XML element, where the name of the property is the name of the XML element, and the value of the property MUST be either blank, or a

well-formed XML element value.

2.2.2 Property Namespace

2.2.2.1 Problem Definition

The requirement is to be able to associate a value with a property name on a resource. It must be possible to associate a URL with the property name.

2.2.2.2 Solution Requirement

Ideally a property namespace should work well with extant property implementations as well as database systems. The DAV property namespace has been specified with the following two facts in mind:

- Namespaces associated with flat file systems are ubiquitous.
- The majority of databases use a fixed schema mechanism.

The last point makes efficient implementation of hierarchical properties difficult. Specifically, each property has a random set of children; the best a relational database can do is provide a table with name and value, where the value is a series of indexes into other tables and each index represents a specific value. However most RDBS do not provide for table pointers, only index values. Such a system would have to be jury-rigged to handle table pointers. In addition, indexing systems are optimized for a small set of relatively large tables; hierarchical property systems tend toward many properties, each with different numbers and types of children, thus potentially requiring a table for each child.

It would seem best to implement a flat property namespace, inducing a natural isomorphism between DAV and most native file systems. Adopting such a model will not restrict RDBS from taking full advantage of their search facilities.

However, it seems that future trends might be toward hierarchical properties. Therefore, DAV requirements [Slein et al.] stipulate that the design of the flat property system MUST be such that it will be possible to add true hierarchical properties later without breaking downlevel clients. Specifically, a flat client MUST be able to speak to a hierarchical server and a hierarchical client MUST be able to speak to a flat server. Worst case either way MUST be that the request fails.

2.2.2.3 Property Names

A property name identifies both the syntax and semantics of the

property's value. It is critical that property names do not collide, e.g., two principals defining the same property name with two different meanings.

The URI framework provides a mechanism to prevent namespace

collision and for varying degrees of administrative control. Rather than reinvent these desirable features, DAV properties make use of them by requiring that all DAV property names MUST be URIs. Since a property is also an XML element, the name of the XML element is a URI.

The property namespace is flat, that is, it is not possible to string together a series of property names in order to refer to a hierarchy of properties. Thus it is possible to refer to a property B but not a property A/B, where A is also a property defined on the resource.

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

2.3 Schemas

A schema is a group of property names and XML elements.

Schema discovery is used to determine if a system supports a group of properties or XML elements. A property does not necessarily contain sufficient information to identify any schema(s) to which it may belong.

As with property names, schemas MUST use URIs as their names.

A resource declares its support for a schema by defining a property whose name is the same as the schema's. The property SHOULD contain the PropSchema XML element.

2.3.1 PropSchema XML Element

Name: http://www.ietf.org/standards/dav/PropSchema
Purpose: To provide information about properties
Schema: http://www.ietf.org/standards/dav/
Parent: Any
Values= [DTD] [DefinedProps]
Description:This property contains the definition of the schema.

This definition consists of two parts. A DTD element that contains a DTD that declares all XML elements and DefinedProps that defines any properties associated with the schema. As with all XML it is possible to add extra XML elements. Therefore schemas may define extra XML elements which are to be included with their values.

2.3.2 DTD XML Element

Name: http://www.ietf.org/standards/dav/DTD
Purpose: To contain the DTD for XML elements associated with the schema.
Schema: http://www.ietf.org/standards/dav/
Parent: Any
Values: XML Declaration statements

2.3.3 DefinedProps XML Element

Name: http://www.ietf.org/standards/dav/DefinedProps
Purpose: To contain a list of properties defined by the schema.
Schema: http://www.ietf.org/standards/dav/
Parent: Any
Values= 1*PropEntries

2.3.4 PropEntries XML Element

Name: http://www.ietf.org/standards/dav/PropEntries
Purpose: To contain the name of a defined property, the DTD of its value, and its live/dead status.
Schema: http://www.ietf.org/standards/dav/
Parent: DefinedProps
Values= Prop [DTD] [Live]
Description:Prop contains the name of the property. The DTD contains the DTD of the property's value. Live, if defined, indicates that the property has semantics and syntax that are enforced by the server.

2.3.5 Live XML Element

Name: http://www.ietf.org/standards/dav/Live
Purpose: If present this indicates the server MUST enforce the syntax and semantics of the property.
Schema: http://www.ietf.org/standards/dav/

Parent: PropEntries

2.4 DAV Schema

The DAV Schema is specified as http://www.ietf.org/standards/dav/. This schema is used to indicate support for

- properties that may be defined on a resource and
- XML elements that may be returned in responses.

2.4.1 DAV Property

Name: http://www.ietf.org/standards/dav
Purpose: Defines support for the DAV schema and protocol.
Schema: http://www.ietf.org/standards/dav/
Values= PropSchema Level
Description:This property indicates that the resource supports the DAV schema and protocol to the level indicated. THE VALUE IN PROPSCHEMA IS TBD, WE NEED TO PROVIDE IT IN AN APPENDIX.

2.4.2 Level XML Element

Name: http://www.ietf.org/standards/dav/level
Purpose: To indicate the level of DAV compliance the resource meets.
Schema: http://www.ietf.org/standards/dav/
Parent: DAV
Values= "1" | "2" | "3"
Description:A value of 1 for level indicates that the resource supports the property and namespace sections of the DAV specification. Level 2 indicates that the resource supports level **1 and the lock section of the specification, with a minimum** locking capability of the write lock. Level 3 indicates support for levels 1 and 2 as well as support for the versioning section of the DAV specification.

2.4.3 Prop XML element

Name: http://www.ietf.org/standards/dav/prop

Purpose: Contains properties related to a resource.
Schema: http://www.ietf.org/standards/dav/
Parent: Any

Values: XML Elements
Description:The Prop XML element is a generic container for properties defined on resources. All elements inside Prop MUST define properties related to the resource. No other elements may be used inside of a Prop element.

2.4.4 PropLoc XML Attribute

Name: http://www.ietf.org/standards/dav/PropLoc
Purpose: To specify the location of the associated property.
Schema: http://www.ietf.org/standards/dav/
Values= URL
Description:This attribute is used with elements inside of Props contained in responses to specify the URL of the property on the associated resource. The PropLoc attribute MUST NOT be used in requests.

2.4.5 Example

```
<?XML:Namespace href="http://www.ietf.org/standards/dav/"
AS="D"/>
<?XML:Namespace href="AIIM:Dublin:" AS="A"/>
<D:Prop>
     <A:Author
D:PropLoc="http://www.foo.com/resource/props/Author">
          Larry Masinter
     </A:Author>
</D:Prop>
```

The previous specifies that the property author exists on some unspecified resource and that the property can be directly referenced at http://www.foo.com/resource/props/Author. The resource upon which the property is defined must be determined from context.

2.5 Property Identifiers

2.5.1 Problem Definition

DAV properties are resources and thus may have a URI where the value of an instance of the property may be retrieved. This URI is separate from the URI name of the property, which identifies the syntax and semantics of the property, but which does not give information on how to access the value of an instance of the property.

A server is free to assign whatever URI it chooses to identify an instance of a property defined on a resource. In fact, a server

is free not to reveal the URI of an instance of a particular resource and instead require that the client access the property through PROPFIND and PROPPATCH. However, many servers will want to allow clients to directly manipulate properties. On these servers, a client can discover the URI of an instance of a property by performing a PROPFIND and examining the PropLoc attribute, if returned, of each property.

2.6 Link XML Element

2.6.1 Problem Description

A mechanism is needed to associate resources with other resources. These associations, known as links, consist of three values, a type describing the nature of the association, the source of the link, and the destination of the link. In the case of annotation, neither the source nor the destination of a link need be the resource upon which the link is recorded.

2.6.2 Solution Requirements

The association mechanism MUST make use of the DAV property mechanism in order to make the existence of the associations searchable.

2.6.3 Link XML Element

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

2.6.4 Src XML Element

Name: http://www.ietf.org/standards/dav/src
Purpose: To indicate the source of a link.
Schema: http://www.ietf.org/standards/dav/

Parent: http://www.ietf.org/standards/dav/link
Values= URI

2.6.5 Dst XML Element

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

2.6.6 Example

```
<?XML:Namespace
     href = "http://www.ietf.org/standards/dav/" AS = "D"/>
<?XML:Namespace
     href = "http://www.foocorp.com/Project/" AS = "F"/>
<D:Prop>
     <Source>
          <Link>
               <F:ProjFiles>Source</F:ProjFiles>
               <src>http://foo.bar/program</src>
               <dst>http://foo.bar/src/main.c</dst>
          </Link>
          <Link>
               <F:ProjFiles>Library</F:ProjFiles>




               <src>http://foo.bar/program</src>
               <dst>http://foo.bar/src/main.lib</dst>
          </Link>
          <Link>
               <F:ProjFiles>Makefile</F:ProjFiles>
               <src>http://foo.bar/program</src>
               <dst>http://foo.bar/src/makefile</dst>
     <Link>
     </Source>
</D:Prop>
```

In this example the resource http://foo.bar/program has a source property defined which 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.

2.7 Multi-Status Response

2.7.1 Problem Definition

Some methods effect more than one resource. The effect of the method on each of the scoped resources may be different, as such a return format that can specify the effect of the method on each resource is needed.

2.7.2 Solution Requirements

The solution must:
- communicate the status code and reason
- give the URI of the resource on which the method was invoked
- be consistent with other return body formats

2.7.3 Multi-Status Response

The default multi-status response body is an text/xml HTTP entity that contains a single XML element called multiresponse, which contains a set of XML elements called response, one for each 200, 300, 400, and 500 series status code generated during the method invocation. 100 series status codes MUST NOT be recorded in a response XML element.

2.7.3.1 MultiResponse

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

2.7.3.2 Response

Name: http://www.ietf.org/standards/dav/response
Purpose: Holds a single response
Schema: http://www.ietf.org/standards/dav/
Parent: Any
Value: (Prop | HREF) Status [ResponseDescription]
Description: Prop MUST contain one or more empty XML elements representing the name of properties. Multiple properties may be included if the same response applies to them all. If HREF is used then the response refers to a problem with the referenced resource, not a property.

2.7.3.3 Status

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

2.7.3.4 ResponseDescription

Name:
http://www.ietf.org/standards/dav/ResponseDescription
Purpose: Contains a message that can be displayed to the user explaining the nature of the response.
Schema: http://www.ietf.org/standards/dav/
Parent: Multiresponse and/or Response
Value: Any
Description: This XML element provides information suitable to be presented to a user.

2.8 Properties and Methods

2.8.1 DELETE

As properties are resources, the deletion of a property causes the same result as the deletion of any resource. It is worth pointing out that the deletion of a property effects both direct manipulation, that is by the property's URL, as well as indirect discovery and manipulation, that is PROPPATCH and PROPFIND.

2.8.2 GET

A GET with a Request-URI that identifies a property returns the name and value of that property. Accept types may be used to specify the format of the return value, but all DAV compliant

servers MUST at minimum support a return type of text/xml. If text/xml is used as the response format then it MUST return the name and value of the property using the Prop XML element.

2.8.2.1 Example

The following example assumes that the property's URL, originally generated by the server, was discovered by examining the proploc XML attribute returned on a result from a FINDPROP.

```
GET /bar.html;prop=z39.50_authors HTTP/1.1
Host: foo.com

HTTP/1.1 200 OK
Content-Type: text/xml
Content-Length: xxxx
E-tag: "1234"
Last-Modified: xxxx

<?XML:Namespace
     href = "http://www.ietf.org/standards/dav/" AS = "D"/>
<?XML:Namespace
     href = "http://www.w3.com/standards/z39.50/"AS = "Z"/>
<D:prop>
     <Z:Authors>
          <Z:Author>Jane Doe</Z:Author>
          <Z:Author>Joe Doe</Z:Author>
           <Z:Author>Lots o'Doe</Z:Author>
     </Z:Authors>
</D:prop>
```

2.8.3 PROPPATCH

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

All DAV compliant servers MUST process instructions which are specified using the PropertyUpdate, Create, and Remove XML elements of the DAV schema. The request message body MUST contain at least one PropertyUpdate XML element. Instruction processing MUST occur in the order instructions are received (i.e., from top to bottom), and MUST be performed atomically.

2.8.3.1 PropertyUpdate XML element

Name: http://www.ietf.org/standards/dav/PropertyUpdate
Purpose: To contain a request to alter the properties on a resource.
Schema: http://www.ietf.org/standards/dav/
Parent: Any
Values= *(Create | Remove)
Description:This XML element is a container for the information required to modify the properties on the resource. This XML element is multi-valued.

2.8.3.2 Create XML element

Name: http://www.ietf.org/standards/dav/create
Purpose: To create the DAV properties specified inside the Create XML element.
Schema: http://www.ietf.org/standards/dav/
Parent: http://www.ietf.org/standards/dav/PropertyUpdate
Values= Prop
Description:This XML element MUST contain only a Prop XML element. The elements contained by Prop specify the name and value of properties that are created on Request-URI. If a property already exists then its value is replaced. The Prop XML element MUST NOT contain a PropLoc XML attribute.

2.8.3.3 Remove XML element

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

2.8.3.4 Response

The response MUST have a response body that contains a

multiresponse identifying the results for each property.

2.8.3.5 Response Codes

200 OK - The command succeeded. As there can be a mixture of Create and Removes in a body, a 201 Create seems inappropriate.
403 Forbidden - The client, for reasons the server chooses not to specify, can not alter one of the properties.
405 Conflict - The client has provided a value whose semantics are not appropriate for the property. This includes trying to set read only properties.
413 Request Entity Too Long - If a particular property is too long to be recorded then a composite XML error will be returned indicating the offending property.
417 Insufficient Space on Resource - The resource does not have sufficient space to record the state of the resource after the execution of this method.
418 Atomicity Failure - The command was not executed because of an atomicity failure elsewhere the caused the entire command to be aborted.

2.8.3.6 Example

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

<?XML:Namespace
     href = "http://www.ietf.org/standards/dav/" AS = "D"/>
<?XML:Namespace
     href = "http://www.w3.com/standards/z39.50/" AS = "Z"/>
<D:PropertyUpdate>
     <Create>
          <prop>
               <Z:authors>
                    <Z:Author>Jim Whitehead</Z:Author>
                    <Z:Author>Roy Fielding</Z:Author>
               </Z:authors>
          </Prop>
     </Create>
     <Remove>
          <prop><Z:Copyright-Owner/></prop>
     </Remove>
</D:PropertyUpdate>
```

```
HTTP/1.1 405 Conflict
Content-Type: text/xml
Content-Length: xxxxx

<?XML:Namespace
     href="http://www.ietf.org/standards/dav/" AS = "D"/>
<?XML:Namespace
     href="http://www.w3.com/standards/z39.50/" AS = "Z"/>
<D:MultiResponse>
     <ResponseDescription> Copyright Owner can not be deleted or
altered.</ResponseDescription>
     <Response>
          <Prop><Z:authors/></Prop>
          <Status>HTTP/1.1 418 Atomicity Failure</Status>
     </Response>
     <Response>
          <Prop><Z:Copyright-Owner/></Prop>
          <Status>HTTP/1.1 405 Conflict</Status>
     </Response>
</D:MultiResponse>
```

2.8.4 PUT

A PUT is specified in order to control what is returned by a GET. However a GET on a property always returns a predefined property containment format. Therefore PUT can not be used if the Request-URI refers to a property.

2.8.5 PROPFIND

The PROPFIND method retrieves properties defined on Request-URI. The request message body is an XML document that MUST contain only one PropFind XML element, which specifies the type of property find action to be performed. The XML element contained by PropFind specifies the type of action to be performed: retrieve all property names and values (AllProp), retrieve only specified property names and values (Prop), or retrieve only a list of all property names (Propname). When a Prop XML element is present, it specifies a list of the names of properties whose name and value are to be returned. The Prop element, when used within a FINDPROP request body MUST be empty.

The response is a text/xml message body that contains a MultiResponse XML element which describes the results of the attempts to retrieve the various properties. If a property was successfully retrieved then its value MUST be returned in the prop XML element. In the case of Allprop and Findprop, if a principal does not have the right to know if a particular

property exists, an error MUST NOT be returned. The results of this method SHOULD NOT be cached.

2.8.5.1 Propfind XML element

Name: http://www.ietf.org/standards/dav/Propfind
Purpose: To specify the set of matching properties
Schema: http://www.ietf.org/standards/dav/
Parent: Any
Values= (Prop | Allprop | Propname)
Description: Propfind is a container element for the exact specification of a PROPFIND request.

2.8.5.2 Allprop

Name: http://www.ietf.org/standards/dav/Allprop
Purpose: To specify that all properties are to be returned
Schema: http://www.ietf.org/standards/dav/
Parent: Propfind
Description: Its presence in a PROPFIND request specifies the name and value of all properties defined on the resource MUST be returned.

2.8.5.3 Propname

Name: http://www.ietf.org/standards/dav/Propname
Purpose: To specify that the names of all properties defined on the resource are to be returned.
Schema: http://www.ietf.org/standards/dav/
Parent: Propfind
Description: Its presence in a PROPFIND request specifies the name of all properties defined on the resource MUST be returned.

2.8.5.4 PropFindResult XML element

Name: http://www.ietf.org/standards/dav/PropFindResult
Purpose: To contain the results of a SEARCH request
Schema: http://www.ietf.org/standards/dav/
Parent: Any
Values: Prop

2.8.5.5 Example 1 - Prop

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


<?XML:Namespace href =
     "http://www.ietf.org/standards/dav/" AS = "G"/>
<?XML:Namespace href =
     "http://www.foo.bar/boxschema/" AS = "B"/>
<G:PROPFIND>
     <prop>
          <B:bigbox>
          <B:author>
          <B:DingALing>
          <B:Random>
     </prop>
</G:PROPFIND>

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

<?XML:Namespace
     href ="http://www.ietf.org/standards/dav/" AS = "S">
<?XML:Namespace href = "http://www.foo.bar/boxschema" AS = R">
<D:MultiResponse>
     <ResponseDescription> There has been an access violation
error. </ResponseDescription>
     <Response>
          <Prop>
               <R:bigbox D:Proploc="http://prop.com/BoxType">




                    <BoxType>Box type A</BoxType>
               </R:bigbox>
               <R:author D:Proploc="http://prop.com/Author">
                    <Name>J.J. Dingleheimerschmidt</Name>
               </R:author>
          </Prop>
          <Status>HTTP/1.1 200 Success</Status>
</Response>
     <Response>
          <Prop><R:DingALing/><R:Random/></>
          <Status>HTTP/1.1 403 Forbidden</Status>
          <ResponseDescription> The user does not have access to
the DingALink property. </ResponseDescription>
     </Response>
```

```
</D:MultiResponse>
```

The result will return all properties on the container. In this case only two properties were found. The principal did not have sufficient access rights to see the third and fourth properties so an error was returned.

2.8.5.6 Example 2 - Allprop

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

<?XML:Namespace href =
     "http://www.ietf.org/standards/dav/" AS = "G"/>
<G:PROPFIND>
     <Allprop/>
</G:PROPFIND>

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

<?XML:Namespace href =
     "http://www.ietf.org/standards/dav/" As = "S">
<?XML:Namespace href = "http://www.foo.bar/boxschema" AS = R">
<S:MultiResponse>
     <Prop>
          <R:bigbox D:Proploc="http://prop.com/BigBox">
               <BoxType>Box type A</BoxType>
          </R:bigbox>
          <R:author D:Proploc="http://prop.com/Author">
               <Name>Hadrian</Name>
          </R:author>
     </Prop>
     <Status>HTTP/1.1 200 Success</Status>
</S:MultiResponse>
```

This particular client only had the right to see two properties, BoxType and Author. No error is returned for the remaining properties, as the client does not even have sufficient rights to know they exist. If the client did have the right to know they existed but did not have the right to see their value, a 207 multi-response with a multiresponse, as used in the previous example, would have been returned.

2.8.5.7 Example 3 - Propname

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

<?XML:Namespace
     href = "http://www.ietf.org/standards/dav/" AS = "G"/>
<G:PROPFIND>
     <Propname/>
</G:PROPFIND>

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

<?XML:Namespace
     href = "http://www.ietf.org/standards/dav/" As = "S">
<?XML:Namespace
     href = "http://www.foo.bar/boxschema" AS = "R">
<S:MultiResponse>
     <Prop>
          <R:bigbox D:Proploc="http://prop.com/BigBox"/>
          <R:author D:Proploc="http://prop.com/Author"/>
          <R:DingALing/>
          <R:Random/>
     </Prop>
     <Status>HTTP/1.1 200 Success</Status>
</S:MultiResponse>
```

In this case only two of the properties have direct URLs available, while the other two properties can only be referenced via PROPFIND and PROPPATCH.

3 A Proposal for Collections of Web Resources and Name Space Operations

3.1 Observations on the HTTP Object Model

This section provides a description of a new type of Web resource, the collection, and discusses its interactions with the HTTP URL namespace. This discussion is a prerequisite for the specification of methods that operate on collections, given later in this document.

3.1.1 Collection Resources

A collection is a resource whose state consists of a list of internal members, a list of external members, and a set of properties. An internal member resource MUST have a URI that is immediately relative to the base URI of the collection, that is, a relative URI in which "../" is illegal, which must begin with "./" and which MAY contain only one other "/" at the end of the URI. An external member resource MUST be an absolute URI that is not an internal URI. Any given internal or external URI MUST only belong to the collection once, i.e., multiple instances of URIs in a collection are illegal. Properties defined on collections have no special distinction, and behave exactly as do properties on non-collection resources.

The purpose of a collection resource is to model collection-like objects (e.g., a filesystem directory) within a server's namespace. Once these objects have been modeled with collections, a client may perform an INDEX, add and remove

external members using ADDREF and DELREF, and perform recursive operations, such as a full hierarchy copy.

To support methods which operate on collections, a server SHOULD model its collection-like objects with collection resources. For example, a server which is implemented on top of a filesystem SHOULD treat all filesystem directories exposed by the server as collection resources.

3.1.2 Creation and Retrieval of Collection Resources

This document specifies the MKCOL method to create new collection resources, and the INDEX method to list their contents.

In HTTP/1.1, the PUT method is defined to store the request body at the location specified by Request-URI. While a description format for a collection can readily be constructed that could be used 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 if they both use the same method.

While it might seem desirable to have GET return a listing of the members of a collection, this is foiled by the existence of the "index.html" de-facto standard namespace redirection, in which a GET request on a collection is automatically redirected to the index.html resource.

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

3.1.2.1 Example

The structured resource http://foo/bar is created with a PUT. Bar is a multipart/related file with two members http://foo/bar/a and http://foo/bar/b. If bar were deleted then both a and b would also be deleted since they are all really just one resource. If http://foo/bar/a/c was PUT then a DELETE on http://foo/bar/a would also delete http://foo/bar/a/c as c was created with a PUT not a MKCOL.

If http://foo/bar/b/d is created with a MKCOL and http://foo/bar/b/d/e was created then a DELETE on d would fail because d is a collection with an internal member. However the existence of the collection d is something of an illusion. If a DELETE was executed on http://foo/bar then everything would be deleted, even though http://foo/bar/b/d was created with a MKCOL.

Thus the effect of a MKCOL within a composite resource s namespace is felt on its children, not its ancestors. The children of d MUST be treated as members of a collection when a method is executed on d. But a method executed on b or a is treated as if there only existed a non-collection resource.

3.1.3 Source Resources and Output Resources

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

server prior to transmission.

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

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

In the absence of distributed authoring capability, it is acceptable to have no mapping of source resource(s) to the URI namespace, and in fact 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.

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

3.2 MKCOL Method

3.2.1 Problem Description

A client must be able to create a collection.

3.2.2 Solution Requirements

The solution must ensure that a collection has been made (i.e. that it responds to the INDEX method) as opposed to a non-

collection resource. If a collection could not be made, it must indicate this failure to the user-agent.

3.2.3 Request

MKCOL creates a new collection resource at the location specified by the Request-URI. If the Request-URI exists, then MKCOL must fail. During MKCOL processing, a server MUST make the Request-URI a member of its parent collection. If no such an 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/ exist, the request MUST fail.

3.2.3.1 MKCOL Without Request Body

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

3.2.3.2 MKCOL With Request Body

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.

3.2.4 Response

Responses from a MKCOL request are not cacheable, since MKCOL has non-idempotent semantics.
201 (Created) - The collection or structured resource was created in its entirety.
403 (Forbidden) - This indicates at least one of two conditions: 1) The server does not allow the creation of collections at the

given location in its namespace, and 2) The parent collection of the Request-URI exists but cannot accept members.
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.
417 (Insufficient Space on Resource) - The resource does not have sufficient space to record the state of the resource after the execution of this method.

3.2.5 Example

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

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


HTTP/1.1 201 Created
```

3.3 Standard DAV Properties

The following properties are defined on DAV compliant resources. All enclosed properties are part of the DAV Schema.

3.3.1 IsCollection Property

Name: http://www.ietf.org/standards/dav/iscollection
Purpose: This property contains a Boolean value that is set to "true" if the resource is a collection
Schema: http://www.ietf.org/standards/dav/
Value: ("true" | "false")
Description: This property MUST be defined on all DAV compliant resources.

3.3.2 DisplayName Property

Name: http://www.ietf.org/standards/dav/displayname
Purpose: A name for the resource that is suitable for presentation to a user.
Schema: http://www.ietf.org/standards/dav/
Value: Any valid XML character data (as defined in [Bray, Sperberg-McQueen, 1997])

Description: This property SHOULD be defined on all DAV compliant resources. If present, the property contains a description of the resource that is suitable for presentation to a user.

3.3.3 CreationDate Property

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

3.3.4 GETentity Property

Name: http://www.ietf.org/standards/dav/GETentity
Purpose: Contains the value of headers that are returned by a GET without Accept headers.
Schema: http://www.ietf.org/standards/dav/
Value: Content-Type Content-Length Content-Language Last-Modified Etag Creation-Date
Description: This property MUST be defined on all DAV compliant resources unless GET is not supported, in which case this property MUST NOT be defined. This property MUST contain at most one instance of each element in its Value, if they are defined.

3.3.5 INDEXentity Property

Name: http://www.ietf.org/standards/dav/INDEXentity
Purpose: Contains the value of headers that are returned by an INDEX.
Schema: http://www.ietf.org/standards/dav/
Value: Content-Type Content-Length Content-Language Last-Modified Etag Creation-Date

Description: This property MUST be defined on all DAV compliant resources unless INDEX is not supported, in which case this property MUST NOT be defined. This property MUST contain at most one instance of each element in its Value, if they are defined.

3.3.6 Content-Type XML Element

Name: http://www.ietf.org/standards/dav/content-type
Purpose: The content-type of the member resource.
Schema: http://www.ietf.org/standards/dav/
Parent: GETentity or INDEXentity
Value: media-type ; defined in Section 3.7 of [Fielding et al., 1997]
Description: If the parent of this element is GETentity, the value MUST be identical to the content-type returned by a GET on the resource without Accept headers. If the parent is INDEXentity, the value MUST be identical to the content-type returned by an INDEX on the resource. If no content-type is available, this element MUST NOT be defined.

3.3.7 Content-Length XML Element

Name: http://www.ietf.org/standards/dav/content-length
Purpose: Describes the default content-length of the resource.
Schema: http://www.ietf.org/standards/dav/
Value: content-length ; see section 14.14 of RFC 2068
Description: If the parent of this element is GETentity, this element MUST have a value equal to the content-length header returned by a GET on the resource without Accept headers. If the parent is INDEXentity, the value MUST be identical to the content-length returned by an INDEX on the resource. If no content-length is available, this element MUST NOT be defined.

3.3.8 Content-Language XML Element

Name: http://www.ietf.org/standards/dav/content-language
Purpose: Describes the default natural language of a resource.
Schema: http://www.ietf.org/standards/dav/
Value: language-tag ;language-tag is defined in section 14.13 of RFC 2068
Description: If the parent of this element is GETentity, this element MUST have a value equal to the content-language header returned by a GET on the resource without Accept headers. If the parent is INDEXentity, the value MUST be identical to the content-language header returned by an INDEX on the resource. If no content-language header is available, this element MUST NOT be defined.

3.3.9 Last-Modified XML Element

Name: http://www.ietf.org/standards/dav/last-modified
Purpose: The date the resource was last modified.
Schema: http://www.ietf.org/standards/dav/
Parent: GETentity or INDEXentity
Value: The date MUST be given in RFC 1123 format using the

rfc-1123 production, defined in section 3.3.1 of [Fielding et al., 1997].
Description: If the parent of this element is GETentity, this element MUST have a value equal to the last-modified header returned by a GET on the resource without Accept headers. If the parent is INDEXentity, the value MUST be identical to the last-modified header returned by an INDEX on the resource. If no last-modified header is available, this element MUST NOT be defined.

3.3.10 Etag XML Element

Name: http://www.ietf.org/standards/dav/etag
Purpose: The entity tag of the resource.
Schema: http://www.ietf.org/standards/dav/
Parent: GETentity or INDEXentity
Value: entity-tag ; defined in Section 3.11 of [Fielding et al., 1997]
Description: If the parent of this element is GETentity, this element MUST have a value equal to the entity-tag header returned by a GET on the resource without Accept headers. If the parent is INDEXentity, the value MUST be identical to the entity-tag header returned by an INDEX on the resource. If no entity-tag header is available, this element MUST NOT be defined.

3.4 INDEX Method

3.4.1 Problem Description

A mechanism is needed to discover if a resource is a collection and if so, list its members.

3.4.2 Solution Requirements

The solution:
- must allow a client to discover the members of a collection
- must always provide a machine-readable description of the membership of a collection
- must be leveraged as a more general mechanism to provide a list of contents for any resource which can profitably return a membership like listing.

3.4.3 The Request

The INDEX method returns a machine-readable representation of the membership of the resource at the Request-URI.

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

Collections that respond to an INDEX method with a text/xml entity MUST contain only one ResInfo element. This ResInfo element contains an Href element, which gives the identifier(s) of the resource, a Prop element, which gives selected properties of the resource, and a Members element, which contains a ResInfo element for each member of the collection. The Prop element MUST

contain at least the following properties, if they are defined and available: DisplayName, IsCollection, CreationDate, GETentity, and INDEXentity.

The response from INDEX is cacheable, and SHOULD be accompanied by an ETag header (see section 13.3.4 of RFC 2068). If GET and INDEX return different entities for the same resource state, they MUST return different entity tags.

3.4.4 The Response

200 (OK) - The server MUST send a machine readable response entity which describes the membership of the resource.

3.4.5 ResInfo XML Element

Name: http://www.ietf.org/standards/dav/resinfo
Purpose: Describes a resource.
Schema: http://www.ietf.org/standards/dav/
Parent: Any
Value: Href Prop Members
Description: There MUST be at least one Href element. Each Href element contains a URI for the resource, which MUST be an absolute URI. There MUST be a single Prop element that contains a series of properties defined on the resource. If the resource is a collection, it MAY have at most one Members element, which describes the members of the collection.

3.4.6 Members XML Element

Name: http://www.ietf.org/standards/dav/members
Purpose: Describes the membership of a collection resource.
Schema: http://www.ietf.org/standards/dav/
Parent: ResInfo
Value: ResInfo
Description: Contains zero or more ResInfo elements, which describe members of the collection.

3.4.7 Href XML Element

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

3.4.8 Example

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

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

<?XML:Namespace
     href = "http://www.ietf.org/standards/dav/" as = "D"/>




<D:ResInfo>
     <XML:Href>
          http://www.microsoft.com/user/yarong/dav_drafts/
     </XML:Href>
     <Prop>
        <DisplayName>
          WebDAV working drafts directory
        </DisplayName>
        <IsCollection>true</IsCollection>
        <CreationDate>19970418T070304Z</CreationDate>
        <GETentity>
          <Content-Type>text/html</Content-Type>
```

```
          <Content-Length>2754</Content-Length>
          <Content-Language>en</Content-Language>
          <Last-Modified>
               Fri, 22 Aug 1997 10:11:26 GMT
          </Last-Modified>
          <Etag>"8675309"</Etag>
        </GETentity>
        <INDEXentity>
          <Content-Type>text/xml</Content-Type>
          <Content-Length>xxx</Content-Length>
          <Last-Modified>
               Thu, 11 Sep 1997 23:45:12 GMT
          </Last-Modified>
          <Etag>"fooyyybar"</Etag>
        </INDEXentity>
      </Prop>

      <Members>
        <ResInfo>
          <XML:Href>
          http://www.microsoft.com/user/yarong/dav_drafts/base
          </XML:Href>
          <Prop>
             <IsCollection
      D:PropLoc="http://www.microsoft.com/user/yarong/dav_drafts/b
      ase;props=IsCollection">
               False
             </IsCollection>
             <DisplayName>
               WebDAV Name Space Operations Draft
             </DisplayName>
             <Creation-Date>19970320T230525Z</Creation-Date>

             <GETentity>
               <Content-Type>application/msword</Content-Type>
               <Content-Length>1400</Content-Length>
               <Content-Language>en</Content-Language>
               <Last-Modified>
                    Fri, 22 Aug 1997 18:22:56 GMT
               </Last-Modified>
               <Etag>"8675309"</Etag>
             </GETentity>
          </Prop>
        </ResInfo>
      </Members>
</D:ResInfo>
```

This example shows the result of the INDEX method applied to the collection resource http://www.microsoft.com/user/yarong/dav_drafts/. It returns a response body in XML format, which gives information about the

container and its sole member, http://www.microsoft.com/user/yarong/dav_drafts/base. The entry

on the collection confirms that the resource the INDEX was executed on is a collection. The result also contains the URI of the IsCollection property on the member resource.

3.5 Behavior of RFC 2068 Methods on Collections

With the introduction of the collection resource type to the HTTP object model, it is necessary to define the behavior of the existing methods (defined in RFC 2068) when invoked on a collection resource to avoid ambiguity. While some methods, such as OPTIONS and TRACE behave identically when applied to collections, GET, HEAD, POST, PUT, and DELETE require some additional explanation.

3.5.1 GET, HEAD for Collections

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

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

3.5.2 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.

3.5.3 PUT for Collections

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

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

3.5.3.1 PUT for Non-Collection Resources

A PUT performed on an existing resource replaces the GET response entity of the resource, but MUST NOT change the value of any dead properties defined on the resource. Live properties defined on

the resource MAY be recomputed during PUT processing.

3.5.4 DELETE for Collections

When DELETE is applied to a collection without internal members the collection resource, along with its properties, and external members, MUST be deleted. A DELETE method applied to a collection resource containing internal member resources MUST fail with a 409 Conflict status code.

3.5.5 DELETE Method for Non-Collection Resources

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

3.6 COPY Method

3.6.1 Problem Description

Currently, in order to create a copy of a resource, the client must GET an entity and then PUT that entity to the desired destination. This requires (1) an entity to be transmitted to and from the server and (2) that the resource be expressible as an entity with complete fidelity.

This is problematic because of the network traffic involved in making a copy, and because there is often no way to fully express a resource as an entity without a loss of fidelity.

3.6.2 Solution Requirements

The solution:
- MUST allow a principal to create a copy of a resource without having to transmit the resource to and from the server.

3.6.3 The Request

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

3.6.3.1 COPY for HTTP/1.1 resources

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

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

3.6.3.2 COPY for Properties

The following section defines how properties on a resource are handled during a COPY operation.
Live properties SHOULD be duplicated as identically behaving live

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

If a property on the source already exists on the resource and the overwrite header is set to TRUE then the property at the destination MUST be overwritten with the property from the source. If the overwrite header is false and the previous situation exists then the COPY MUST fail with a 409 Conflict.

3.6.3.3 COPY for Collections

A COPY on a collection causes a new, empty collection resource to be created at the destination with the same properties as the source resource. All properties on the source collection MUST be duplicated on the destination collection, subject to modifying headers, following the definition for copying properties. The new collection MUST NOT contain any members, internal or external.

3.6.3.4 Type Interactions

If the destination resource identifies a property and the source resource is not a property, then the copy SHOULD fail.

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

3.6.4 The Response

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

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

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

417 (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.

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

3.6.5 Examples

3.6.5.1 Overwrite Example

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

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


HTTP/1.1 200 OK
```

3.6.5.2 No Overwrite Example

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

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


HTTP/1.1 412 Precondition Failed
```

3.7 MOVE Method

3.7.1 Problem Description

The move operation on a resource is the logical equivalent of a copy followed by a delete, where the actions are performed atomically. Using RFC 2068 methods only, this procedure could be performed in several steps. First, the client could issue a GET to retrieve a representation of a resource, issue a DELETE to remove the resource from the server, then use PUT to place the resource on the server with a new URI. As is the case for COPY - because of the network traffic involved in making a move, and because there is often no way to fully express a resource as an entity without a loss of fidelity - server move functionality is preferable.

With a WEBDAV server, a principal may accomplish this task by issuing a COPY and then DELETE. Network load decreases, but the server load may still be significant because the server must create a duplicate resource. Were a server to know beforehand that a principal intended to perform COPY and DELETE operations in succession, it could avoid the creation of a duplicate resource.

3.7.2 Solution Requirements

The solution:
- Must prevent the unneeded transfer of entity bodies from and to the server.
- Must prevent the unneeded creation of copies by the server.

3.7.3 The Request

The move operation on a resource is the logical equivalent of a copy followed by a delete, where the actions are performed atomically. 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.

3.7.4 The Response

200 (OK) - The resource was moved. A successful response must

contain the Content-Location header, set equal to the URI in source. This lets caches properly flush any cached entries for the source. Unfortunately the Content-Location header only allows a single value so it is not possible for caches unfamiliar with the MOVE method to properly clear their caches.

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

501 (Not Implemented) - This may occur if the Destination header specifies a resource which is outside its domain of control (e.g., stored on another server) resource and the server either refuses or is incapable of moving to an external resource.

502 (Bad Gateway) - This may occur when moving to external resources and the destination server refused to accept the resource.

3.7.5 Examples

3.7.5.1 Overwrite Example

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

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


HTTP/1.1 200 OK
Content-Location:
http://www.ics.uci.edu/users/f/fielding/index.html
```

3.8 ADDREF Method

3.8.1 Problem Definition

There needs to be a way to add an external member to a

collection.

3.8.2 Solution Requirements

The solution must:
- allow access control
- allow referencing to URIs of external members
- not require a body

3.8.3 The Request

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

It is not an error if the URI specified in the Collection-Member header already exists as an external member of the collection, however, after processing ADDREF there MUST be only one instance of the URI in the collection. If the URI specified in the Collection-Member header already exists as an internal member of the collection, the ADDREF method MUST fail with a 412 Precondition Failed status code.

3.8.4 Example

```
ADDREF /~whitehead/dav/ HTTP/1.1
HOST: www.ics.udi.edu
Collection-Member: http://www.ietf.org/standards/dav/


HTTP/1.1 200 OK
```

3.9 DELREF Method

3.9.1 Problem Definition

There needs to be a way to remove an external member from a collection.

3.9.2 Solution Requirements

The solution must:
- allow access control
- allow referencing to URIs of external members
- not require a body

3.9.3 The Request

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

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

3.9.4 Example

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

HTTP/1.1 200 OK
```

3.10 PATCH Method

3.10.1 Problem Definition

At present, if a principal wishes to modify a resource, they must issue a GET against the resource, modify their local copy of the resource, and then issue a PUT to place the modified resource on the server. This procedure is inefficient because the entire entity for a resource must be transmitted to and from the server in order to make even small changes. Ideally, the update entity transmitted to the server should be proportional in size to the modifications.

3.10.2 Solution Requirements

The solution must:

- allow partial modification of a resource without having to transmit the entire modified resource
- allow byte-range patching
- allows extensions so that patches can be done beyond simple byte-range patching
- allow ranges to be deleted, inserted, and replaced

3.10.3 The Request

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

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

While server support for PATCH is optional, if a server does support PATCH, it MUST support at least the text/xml diff format defined below. Support for the VTML difference format [VTML] is recommended, but not required.

3.10.4 text/xml elements for PATCH

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

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

3.10.4.1 ResourceUpdate

Name: http://www.ietf.org/standards/dav/patch/resourceupdate
Purpose: Contains an ordered set of changes to a non-collection, non-property resource.
Schema: http://www.ietf.org/standards/dav/patch/
Parent: Any

Value: *(Insert | Delete | Replace)

3.10.4.2 Insert

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

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

3.10.4.3 Delete

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

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

3.10.4.4 Replace

Name: http://www.ietf.org/standards/dav/patch/replace
Purpose: Replaces the specified range of octets with the contents of the XML element. If the number of octets in the XML

element is different from the number of octets specified, the update MUST be rejected.
Schema: http://www.ietf.org/standards/dav/patch/
Parent: ResourceUpdate

Value: The Replace XML element MUST contain an octet-range XML element. The contents of the entity are the replacement octets.

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

3.10.4.5 Octet-Range Attribute

Name: http://www.ietf.org/standards/dav/patch/octet-range
Purpose: Specifies a range of octets that the enclosing property effects.
Schema: http://www.ietf.org/standards/dav/patch/
Parent: Insert, Delete, Replace
Value: number ["-" (number | "end")]
Number = 1*Digit

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

3.10.5 The Response

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

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

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

416 (Unprocessable Entity) - A new status code. The server understands the content type of the request entity, but was unable to process the contained instructions.

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

3.10.6 Examples

3.10.6.1 HTML file modification

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

```
Before:
     <HTML>
     <HEAD>
     <TITLE>Hello world HTML page</TITLE>
     </HEAD>
     <BODY>
     <P>Hello, world!</P>
     </BODY>
     </HTML>
PATCH Request:                        Response:
     PATCH hello.html HTTP/1.1
     Host: www.example.org
     Content-Type: text/xml
     Content-Length: xxx

                                      HTTP/1.1 100 Continue
     <?XML:Namespace href =
     Shttp://www.ietf.org/standards/dav/patch/" AS = "D"/>
     <D:ResourceUpdate>
          <Replace XML-SPACE = "PRESERVE"><octet-range>14</octet-
     range>&003CTITLE&003ENew Title&003C/TITLE&003E</Replace>
          <Delete><octet-range>38-50</Delete>
     <Insert XML-SPACE = "PRESERVE"><octet-range>86</>&
     003CP&003ENew paragraph&003C/P&003E</Insert>
     </D:ResourceUpdate>
                                      HTTP/1.1 200 OK
After:
     <HTML>
     <HEAD>
     <TITLE>New Title</TITLE>
     </HEAD>
     <BODY>
     <P>Hello, world!</P>
     <P>New paragraph</P>
     </BODY>
     </HTML>
```

3.11 Headers

3.11.1 Destination Header

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

```
Destination= "Destination" ":" URI
```

3.11.2 Enforce-Live-Properties Header

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

```
EnforceLiveProperties = "Enforce-Live-Properties" ":" ("*" |
"Omit" | 1#(Property-Name))
Property-Name = "<" URI ">"
```

3.11.3 Overwrite Header

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 "false" states that the server MUST NOT perform the COPY or MOVE operation if the state of the destination resource is non-null. By default, the value of Overwrite is "true," and a client MAY omit this header from a request when its value is "true." 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.

```
Overwrite = "Overwrite" ":" ("true" | "false")
```

If there is a conflict and the Overwrite header equals "true", or is absent and thus defaults to "true", then the method MUST fail with a 409 Conflict.

3.11.4 Destroy Header

When deleting a resource the client often wishes to specify exactly what sort of delete is being enacted. The Destroy header, used with the Mandatory header, allows the client to specify the end result they desire. The Destroy header is specified as follows:

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

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

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

DestroyHeader = "Destroy" ":" #Choices

Choices = "VersionDestroy" | "NoUndelete" | "Undelete" | token |"<" URI ">" ; a token extension MUST NOT be used unless it is specified in a RFC16, otherwise a URI MUST be used for extensions.

3.11.5 Collection-Member Header

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

CollectionMember = "Collection-Member" ":" URI

3.12 Links

3.12.1 Source Link Property Type

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

Schema: http://www.ietf.org/standards/dav/link/
Parent: Any

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

Discussion: The source of the link (src) is typically the URI of the output resource on which the link is defined, and there is typically only one destination (dst) of the link, which is the URI where the unprocessed source of the resource may be accessed. When more than one link destination exists, this specification asserts no policy on ordering.

4 State Tokens

4.1 Overview

4.1.1 Problem Description

There are times when a principal will want to predicate successful execution of a method on the current state of a resource. While HTTP/1.1 provides a mechanism for conditional execution of methods using entity tags via the "If-Match" and "If-None-Match" headers, the mechanism is not sufficiently extensibleto express conditional statements involving more generic state indicators, such as lock tokens.

The fundamental issue with entity tags is that they can only be generated by a resource. However there are times when a client will want to be able to share state tokens between resources, potentially on different servers, as well as be able to generate certain types of lock tokens without first having to communicate with a server.

For example, a principal may wish to require that resource B have a certain state in order for a method to successfully execute on resource A. If the client submits an e-tag from resource B to resource A, then A has no way of knowing that the e-tag is meant to describe resource B.

Another example occurs when a principal wishes to predicate the successful completion of a method on the absence of any locks on a resource. It is not sufficient to submit an "If-None-Match: *" as this refers to the existence of an entity, not of a lock.

This draft defines the term "state token" as an identifier for a state of a resource. The sections below define requirements for state tokens and provide a state token syntax, along with two new headers which can accept the new state token syntax.

4.1.2 Solution Requirements

4.1.2.1 Syntax

Self-Describing. A state token must be self describing such that upon inspecting a state token it is possible to determine what sort of state token it is, what resource(s) it applies to, and what state it represents.

This self-describing nature allows servers to accept tokens from other servers and potentially be able to coordinate state

information cross resource and cross site through standardized protocols. For example, the execution of a request on resource A can be predicated on the state of resource B, where A and B are potentially on different servers.

Client Generable. The state token syntax must allow, when appropriate, for clients to generate a state token without having first communicated with a server.

One drawback of entity tags is that they are set by the server, and there is no interoperable algorithm for calculating an entity tag. Consequently, a client cannot generate an entity tag from a particular state of a resource. However, a state token which encodes an MD5 state hash could be calculated by a client based on a client-held state of a resource, and then submitted to a server in a conditional method invocation.

Another potential use for client generable state tokens is for a client to generate lock tokens with wild card fields, and hence be able to express conditionals such as: "only execute this GET if there are no write locks on this resource."

4.1.2.2 Conditonals

Universal. A solution must be applicable to all requests.
Positive and Negative. Conditional expressions must allow for the expression of both positive and negative state requirements.

4.2 State Token Syntax

State tokens are URLs employing the following syntax:

```
State-Token = "StateToken:" Type ":" Resources ":" State-Info
Type = "Type" "=" Caret-encoded-URL
Resources = "Res" "=" Caret-encoded-URL
Caret-encoded-URL = "^" Resource "^"
```

Resource = <A URI where all "^" characters are escaped>
State-Info = *(uchar | reserved) ; uchar, reserved defined section 3.2.1 of RFC 2068

This proposal has created a new URL scheme for state tokens because a state token names a network resource using its normal name, which is typically state-invariant, along with additional information that specifies a particular state of the resource. Encoding the state information into the native URL scheme of the network resource was not felt to be safe, since freedom from name space collisions could not be guaranteed. If this proposal is accepted, the StateToken URL scheme will need to be defined and registered with IANA.

State Token URLs begin with the URL scheme name "StateToken" rather than the name of the particular state token type they represent in order to make the URL self describing. Thus it is possible to examine the URL and know, at a minimum, that it is a state token.

Labeled name/value pairs are used within the token to allow new fields to be added. Processors of state tokens MUST be prepared to accept the fields in whatever order they are present and MUST ignore any fields they do not understand.
The "Type" field specifies the type of the state information encoded in the state token. A URL is used in order to avoid namespace collisions.

The "Res" field identifies the resource for which the state token

specifies a particular state. Since commas and spaces are acceptable URL characters, a caret is used to delimit a URL. Since a caret is an acceptable URL character, any instances of it must be escaped using the % escape convention.

The State-Info production is expanded upon in descriptions of specific state token types, and is intended to contain the state description information for a particular state token.

4.3 State Token Conditional Headers

4.3.1 If-State-Match

If-State-Match = "If-State-Match" ":" ("AND" | "OR") 1#("<" State-Token ">")

The If-State-Match header is intended to have similar functionality to the If-Match header defined in section 14.25 of RFC 2068.

If the AND keyword is used and all of the state tokens identify the state of the resource, then the server MAY perform the requested method. If the OR keyword is used and any of the state tokens identifies the current state of the resource, then server MAY perform the requested method. If neither of the keyword requirements is met, the server MUST NOT perform the requested method, and MUST return a 412 (Precondition Failed) response.

4.3.2 If-None-State-Match

If-None-State-Match = "If-None-State-Match" ":" 1#("<" State-Token ">")

The If-None-State-Match header is intended to have similar functionality to the If-None-Match header defined in section 14.26 of RFC 2068.

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

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

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

4.4 State Token Header

State-Token-Header = "State-Token" ":" 1#("<" State-Token ">")
The State Token header is intended to have similar functionality to the etag header defined in section 14.20 of RFC 2068. The purpose of the tag is to return state tokens defined on a

resource in a response. The contents of the state-token are not guaranteed to be exhaustive and are generally used to return a new state token that has been defined as the result of a method. For example, if a LOCK method were successfully executed on a resource the response would include a state token header with the lock state token included.

4.5 E-Tags

E-tags have already been deployed using the If-Match and If-None-Match headers. Introducing two mechanisms to express e-tags would only confuse matters, therefore e-tags should continue to be expressed using quoted strings and the If-Match and If-None-Match headers.

5 Locking

5.1 Locking: Introduction

Locking is used to arbitrate access to a resource amongst principals that have equal access rights to that resource.

This specification allows locks to vary over two parameters, the number of principals involved and the type of access to be granted. Furthermore, this document only provides the definition of locking for one access type, write. However, the syntax is extensible, and allows the specification of other access types.

5.1.1 Exclusive Vs. Shared Locks

The most basic form of lock is an exclusive lock. This is a lock where the access right in question is only granted to a single principal. The need for this arbitration results from a desire to avoid having to constantly merge results. In fact, many users so dislike having to merge that they would rather serialize their access to a resource rather than have to constantly perform merges.

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

With shared locks there are two trust sets that affect a resource. The first trust set is created by access permissions. Principals who are trusted, for example, may have permission to write the resource, those who are not, don't. Among those who

have access permission to write the resource, the set of principals who have taken out a shared lock also must trust each other, creating a (typically) smaller trust set within the access permission write set.

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

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

Shared locks are included because experience from web distributed authoring systems has indicated that exclusive write locks are often too rigid. An exclusive write lock is used to enforce a particular editing process: take out exclusive write lock, read the resource, perform edits, write the resource, release the lock. What happens if the lock isn't released? While the time-out mechanism provides one solution, if you need to force the release of a lock immediately, it doesn't help much. Granted, an administrator can release the lock for you, but this could become a significant burden for large sites. In addition there is the problem that an administrator may not be immediately available.

Despite their potential problems, exclusive write locks are extremely useful, since often a guarantee of freedom from overwrite conflicts is what is needed. The tradeoff described in this specification is to provide exclusive write locks, but also to provide a less strict mechanism in the form of shared locks which can be used by a set of people who trust each other and who have access to a communications channel external to HTTP which can be used to negotiate writing to the resource.

5.1.2 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 server implementers have said that they are willing to accept minimum requirements on all services but locking. Locking policy strikes to the very heart of their resource management and versioning systems and they require control over what sort of locking will be made available. For example, some systems 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, the authors are proposing that locking be allowed as the sole axis of negotiation within WebDAV.

5.2 LOCK Method

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

5.2.1 Operation

A LOCK method invocation creates the lock specified by the Lock-Info header on the Request-URI. Lock method requests SHOULD NOT have a request body. A user-agent SHOULD submit an Owner header field with a lock request.

A successful response to a lock invocation MUST include Lock-Token and Time-Out headers.

5.2.2 The Effect of Locks on Properties and Containers

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

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

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

5.2.3 Locking Replicated Resources

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

5.2.4 Locking Multiple Resources

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

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

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

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

If the lock can not be granted to all resources, a 406 Conflict status code MUST be returned with a response entity body containing a multiresponse XML element describing which resource(s) prevented the lock from being granted.

5.2.5 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.

5.2.6 Lock Compatibility Table

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

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

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

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

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

5.2.7 Status Codes

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

412 "Precondition Failed" - The included state-token was not enforceable on this resource or the request in the lock-info header could not be satisfied by the server.

5.2.8 Lock-Info Request Header

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

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

The LockType field specifies the access type of the lock. At present, this specification only defines one lock type, the "Write" lock. The LockScope field specifies whether the lock is

an exclusive lock, or a shared lock. The AddLocks field specifies additional URIs, beyond the Request-URI, to which the lock request applies. The LockTree field is used to specify recursive locks. If the LockTree field is "true", the lock request applies to the hierarchy traversal of the internal members resources of the Request-URI, and the AddLocks URIs, inclusive of the Request-URI and the AddLocks URIs. It is not an error if LockTree is true, and the Request-URI or the AddLocks URIs have no internal member resources. By default, the value of LockTree is "false", and this field MAY be omitted when its value is false.

5.2.9 Owner Request Header

5.2.9.1 Problem Description

When discovering the list of owners of locks on a resource, a principal may want to be able to contact the owner directly. For this to be possible the lock discovery mechanism must provide enough information for the lock owner to be contacted. Additionally, programs may wish to be able to record structured information in the owner header so that other programs may be able to parse it later. Note, however, that these programs may not necessarily be coordinating so there need be no guarantee that the information can be parsed.

5.2.9.2 Solution Requirements

Not all systems have authentication procedures that provide sufficient information to identify a particular user in a way that is meaningful to a person. In addition, many systems that do have sufficient information, such as a name and e-mail address, do not have the ability to associate this information with the lock discovery mechanism. Therefore a means is needed to allow principals to provide authentication in a manner which will be meaningful to a person.

The From header (defined in RFC 2068), which contains only an email mailbox, is not sufficient for the purposes of quick identification. When desperately looking for someone to remove a lock, e-mail is often not sufficient. A telephone number (cell number, pager number, etc.) would be better. Furthermore, the email address in the From header only optionally includes the owners name and that name is often set to an alias anyway. Therefore a header more flexible than From is required.

The value also needs to be such that both man and machine can place values in it and later retrieve those values.

5.2.9.3 Syntax

Owner = "Owner" ":" (Coded-XML | quoted-string)
Coded-XML = field-content ; XML where any character which is not legal in field-content (see section 4.2 of [Fielding et al., 1997]) is XML encoded

The XML SHOULD provide information sufficient for either directly contacting the principal (such as a telephone number or e-mail URI), or for discovering the principal (such as the URL of a homepage) who owns the lock. The quoted string SHOULD provide a means for directly contacting the principal who owns the lock,

such as a name and telephone number.

5.2.10 Time-Out Header

5.2.10.1 Problem Description

In a perfect world principals take out locks, work on the

resource, and then remove the lock when it is no longer needed. However, this process is frequently not completed, leaving active but unused locks. Reasons for this include client programs crashing and losing information about locks, users leaving their systems for the day and forgetting to remove their locks, etc. As a result of this behavior, servers need to establish a policy by which they can remove a lock without input from the lock owner. Once such a policy is instituted, the server also needs a mechanism to inform the principal of the policy.

5.2.10.2 Solution Requirements

There are two basic lock removal policies: administrator and time based removal. In the case of administrator based removal, a principal other than the lock owner has sufficient access rights to order the lock removed, even though they did not take it out. The second policy type is time based removal. In this case, a timer is set as soon as the lock is created. Every time a method is executed on any resource covered by the lock, the timer is reset. If the timer runs out, the lock is removed.

User-agents MUST assume that locks may arbitrarily disappear at any time. If their actions require confirmation of the existence of a lock then the If-State headers are available.

5.2.10.3 Syntax

```
TimeOut = "Time-Out" ":" 1#TimeType
TimeType = ("Second-" DAVTimeOutVal | "Infinite" | Extend)
DAVTimeOutVal = 1*digit
Extend = RFC-Reg | URL "-" Token ; The URL format is used for
unregistered TimeTypes
RFC-Req = Token ; This is a TimeType that has been published as
an RFC
```

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

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

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

submitted in a Time-Out header.

The "Second" TimeType specifies the number of seconds that MUST elapse between granting of the lock at the server, and the

automatic removal of the lock. A server MUST not generate a time out value for "Second" greater than 2^32-1.

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

If the timeout expires then the lock is lost. Specifically the server SHOULD act as if an UNLOCK method was executed by the server on the resource using the lock token of the timed-out lock, performed with its override authority. Thus logs, notifications, and other mechanisms that act as side effects to the granting and removal of a lock will be properly informed as to the disposition of the lock.

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 time-out 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 time-out because its user may be planning on going off-line.

5.2.11 Lock Response

A successful lock response MUST contain a Lock-Token response header, a Time-Out header and a PROP element in the response body which contains the value of the LockDiscovery property.

5.2.11.1 Lock-Token Response Header

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

```
Lock-Token = "Lock-Token" ":" URI
```

5.2.12 Example - Simple Lock Request

```
LOCK /workspace/webdav/proposal.doc HTTP/1.1
Host: webdav.sb.aol.com
Lock-Info: LockType=Write LockScope=Exclusive
Time-Out: Infinite; Second-4100000000
Owner: <?XML:Namespace href="http://www.ietf.org/standards/dav/"
AS =
"D"/><D:HREF>http://www.ics.uci.edu/~ejw/contact.html</D:HREF>


HTTP/1.1 200 OK
Lock-Token: OpaqueLockToken:xyz122393481230912asdfa09s8df09s7df08
sd0f98a098sda
Time-Out: Second-604800
Content-Type: text/xml
Content-Length: xxxxx

<?XML:Namespace
```

```
     href ="http://www.ietf.org/standards/dav/" AS = "D"/>
<D:Prop>
     <lockdiscovery>
          <activelock>
               <locktype>write</locktype>
               <lockscope>exclusive</lockscope>
               <addlocks/>
               <owner>
     <HREF>http://www.ics.uci.edu/~ejw/contact.html</HREF>
               </owner>
               <timeout>Second-604800</timeout>
               <locktoken>
                    <HREF>
               OpaqueLockToken:xyz122393481230912asdfa09s8df09s7d
               f08sd0f98a098sda
                    </HREF>
               </locktoken>
          </activelock>
     </lockdiscovery>
</D:Prop>
```

This example shows the successful creation of an exclusive write lock on resource

http://webdav.sb.aol.com/workspace/webdav/proposal.doc. The resource http://www.ics.uci.edu/~ejw/contact.html contains contact information for the owner of the lock. The server has an activity-based timeout policy in place on this resource, which causes the lock to automatically be removed after 1 week (604800 seconds). The response has a Lock-Token header that gives the state token URL for the lock token generated by this lock request.

5.2.13 Example - Multi-Resource Lock Request

```
LOCK /workspace/webdav/proposal.doc HTTP/1.1
Host: webdav.sb.aol.com
Lock-Info: LockType=Write LockScope=Exclusive
Addlocks=<http://webdav.sb.aol.com/workspace/><http://foo.bar/bla
h>
Time-Out: Infinite, Second-4100000000
Owner: <http://www.ics.uci.edu/~ejw/contact.html>


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

<?XML:Namespace href =
     "http://www.ietf.org/standards/dav/" As = "D"/>
<D:MultiResponse>
     <Response>
          <HREF>
          http://webdav.sb.aol.com/workspace/webdav/proposal.doc
          </HREF>
          <HREF>
          http://webdav.sb.aol.com/workspace/webdav/
          </HREF>
          <Status>HTTP/1.1 202 Accepted</Status>
     </Response>
     <Response>
          <HREF>http://foo.bar/blah</HREF>
          <Status>HTTP/1.1 403 Forbidden</Status>
     </Response>
</D:MultiResponse>
```

This example shows a request for an exclusive write lock on three resources,
http://webdav.sb.aol.com/workspace/webdav/proposal.doc,

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

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

5.3 Write Lock

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

5.3.1 Methods Restricted by Write Locks

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

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

5.3.2 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 to not change while write locked.

If a property is write locked then a LOCK request on the associated resource MUST fail with a 409 "Conflict". Note that a write lock on a property MAY be extended to include the associated resource without the principal having explicitly requested the extension.

5.3.3 Write Locks and Null Resources

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

5.3.4 Write Locks and Collections

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

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

5.3.5 Write Locks and COPY/MOVE

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

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

5.3.6 Re-issuing Write Locks

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

5.3.7 Write Locks and The State-Token Header

5.3.7.1 Problem Definition

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.

5.3.7.2 Solution Requirement

The solution must not require principals to perform discovery in order to prevent accidental overwrites as this could cause race conditions.

The solution must not require that clients guess what sorts of locks might be used and use if-state-match headers with wildcards to prevent collisions. The problem with trying to "guess" which locks are being used is that new lock types might be introduced,

and the program would not know to "guess them". So, for example, a client might put in an if-state-match header with a wildcard specifying that if any write lock is outstanding then the operation should fail. However a new read/write lock could be introduced which the client would not know to put in the header.

5.3.7.3 State-Token Header

The State-Token header, containing a lock token owned by the requesting principal, is used by the principal to indicate that the principal is aware of the existence of the lock specified by the lock token. It is used in the following way.

If the following conditions are met:

1. a user-agent has authenticated itself as a principal,
2. the user-agent is submitting a method request to a resource

on which the principal owns a write lock,

3. the method is restricted by a write lock, as defined in the section "Methods Restricted by a Write Lock",

then the method request MUST include a State-Token header with the lock token of the write lock, or the method fails with a 409 Conflict status code. If multiple resources are involved with a method, such as a COPY or MOVE method, then the lock tokens, if any, for all involved resources, MUST be included in the State-Token request header.

For example, Program A, used by user A, takes out a write lock on a resource. Program A then makes a number of PUT requests on the locked resource, all the requests contain a State-Token header which includes the write lock state token. Program B, also run by User A, then proceeds to perform a PUT to the locked resource. However program B was not aware of the existence of the lock and so does not include the appropriate state-token header. The method is rejected even though principal A is authorized to perform the PUT. Program B can, if it so chooses, now perform lock discovery and obtain the lock token. Note that program A and B can perform GETs without using the state-token header because the ability to perform a GET is not affected by a write lock.

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

5.3.7.3.1 Example

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


HTTP/1.1 200 OK
```

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

5.4 Lock Tokens

5.4.1 Problem Description

It is possible that once a lock has been granted it may be removed without the lock owner s knowledge. This can cause serialization problems if the lock owner executes methods thinking their lock is still active. Due to this, a mechanism is needed for a principal to predicate the successful execution of a message upon the continuing existence of a lock.

5.4.2 Lock Token Introduction

A lock token is a type of state token that describes a particular lock. A lock token is returned by every successful LOCK operation, and can also be discovered through lock discovery on a resource.

There are two types of lock tokens, a generic lock token, which is unique only for a particular resource, and an opaque lock token, which is unique across all resources for all time.

Uniqueness for a particular resource prevents problems with long held outstanding lock tokens being confused with newer tokens. This uniqueness requirement is the same as for e-tags. Uniqueness across all resources for all time allows for tokens to be submitted across resources and servers without fear of confusion.

Generic lock tokens, because of their relaxed uniqueness requirements, are faster to generate than opaque lock tokens.

5.4.3 Generic Lock Tokens

Any valid URI can be used by the resource as a generic lock token. The only requirement is that the lock token MUST never have been issued previously on that resource. Because a lock token is only guaranteed to be unique on the resource that generated it, the lock token MUST NOT be submitted in a state-token request header or an if-state[-not]-match header on any resource but the resource that generated it.

5.4.4 OpaqueLockToken Lock Token

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

one that returned it.

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

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

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

expensive, or they can create a single GUID and then add extension characters. If the second method is selected then the program generating the extensions MUST guarantee that the same extension will never be used twice with the associated GUID.

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

5.5 UNLOCK Method

5.5.1 Problem Definition

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

5.5.2 Example

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

HTTP/1.1 200 OK
```

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

removed from those resources as well.

5.6 Discovery Mechanisms

5.6.1 Lock Capability Discovery

5.6.1.1 Problem Definition

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.

5.6.1.2 SupportedLock Property

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

5.6.1.3 LOCKENTRY XML Element

Name: http://www.ietf.org/standards/dav/lockentry
Purpose: Defines a DAVLockType/LockScope pair which may be legally used with a LOCK on the specified resource.
Schema: http://www.ietf.org/standards/dav/
Parent: A SupportedLock entry
Values: LockType LockScope

5.6.1.4 LOCKTYPE XML Element

Name: http://www.ietf.org/standards/dav/locktype
Purpose: Lists a DAVLockType
Schema: http://www.ietf.org/standards/dav/
Parent: LOCKENTRY
Values: DAVLockTypeValue

5.6.1.5 LOCKSCOPE XML Element

Name: http://www.ietf.org/standards/dav/lockscope
Purpose: Lists a DAVLockScope
Schema: http://www.ietf.org/standards/dav/
Parent: LOCKENTRY
Values: DAVLockScopeValue

5.6.1.6 Example

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

<?XML:Namespace href =
     "http://www.ietf.org/standards/dav/" AS = "D"/>
<D:PROPFIND>
     <prop><SupportedLock/></prop>
</D:PROPFIND>

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

<?XML:Namespace
     href ="http://www.ietf.org/standards/dav/" AS = "D"/>
<D:MultiResponse>
     <Response>
          <Prop>
               <SupportedLock>
                    <LockEntry>
                         <LockType>Write</LockType>
                         <LockScope>Exclusive</LockScope>
                    </LockEntry>
                    <LockEntry>
                         <LockType>Write</LockType>
                         <LockScope>Shared</LockScope>
                    </LockEntry>
```

```
            </SupportedLock>
        </Prop>
        <Status>HTTP/1.1 200 Success</Status>
</Response>
</D:MultiResponse>
```

5.6.2 Active Lock Discovery

5.6.2.1 Problem Definition

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.

5.6.2.2 Solution Requirements

The lock discovery mechanism should provide a list of who has the resource locked, what locks they have, and what their lock tokens are. The lock tokens are useful in shared lock situations where two principals may want to guarantee that they do not overwrite each other. The lock tokens are also useful for administrative purposes so that an administrator can remove a lock by referring to its token.

5.6.2.3 LOCKDISCOVERY Property

Name: http://www.ietf.org/standards/dav/lockdiscovery
Purpose: To discover what locks are active on a resource
Schema: http://www.ietf.org/standards/dav/
Values= An XML document containing zero or more ActiveLock XML elements.

Description: The LOCKDISCOVERY property returns a listing of who has a lock, what type of lock they have, the time out type and the time remaining on the time out, and the associated lock token. The server is free to withhold any or all of this information if the requesting principal does not have sufficient access rights to see the requested data. A server which supports locks MUST provide the LOCKDISCOVERY property on any resource with locks on it.

5.6.2.4 ACTIVELOCK XML Element

Name: http://www.ietf.org/standards/dav/activelock
Purpose: A multivalued XML element that describes a particular active lock on a resource
Schema: http://www.ietf.org/standards/dav/
Parent: A LOCKDISCOVERY entry
Values= LOCKTYPE LOCKSCOPE [ADDLOCKS] OWNER TIMEOUT LOCKTOKEN

5.6.2.5 OWNER XML Element

Name: http://www.ietf.org/standards/dav/lock/owner
Purpose: Returns owner information
Schema: http://www.ietf.org/standards/dav/
Parent: ACTIVELOCK
Values= XML:REF | {any valid XML string}

5.6.2.6 TIMEOUT XML Element

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

5.6.2.7 ADDLOCKS XML Element

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

5.6.2.8 LOCKTOKEN XML Element

Name: http://www.ietf.org/standards/dav/statetoken
Purpose: Returns the lock token
Schema: http://www.ietf.org/standards/dav/
Parent: ACTIVELOCK
Values= HREF
Description: The HREF contains a Lock-Token-URL.

5.6.2.9 Example

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

<?XML:Namespace href =
     "http://www.ietf.org/standards/dav/" AS = "D"/>
<D:PROPFIND>
     <prop><lockdiscovery/></prop>
</D:PROPFIND>

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

<?XML:Namespace
     href ="http://www.ietf.org/standards/dav/" AS = "D"/>
<D:MultiResponse>
     <Response>
          <Prop>
               <lockdiscovery>
                    <activelock>
                         <locktype>write</locktype>
                         <lockscope>exclusive</lockscope>
                         <addlocks>
                              <HREF>http://foo.com/doc/</HREF>
                         </addlocks>
                         <owner>Jane Smith</owner>
                         <timeout>Infinite</timeout>
                         <locktoken>
                              <HREF>iamuri:unique!!!!!</HREF>
                         </locktoken>
                    </activelock>




               </lockdiscovery>
          </Prop>
          <Status>HTTP/1.1 200 Success</Status>
</Response>
</D:MultiResponse>
```

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

6 Version Control

[TBD]

7 Internationalization Support

[TBD]

8 Security Considerations

[TBD]

9 Copyright

10 Acknowledgements

Terry Allen, Harald Alvestrand, Alan Babich, Dylan Barrell, Bernard Chester, Dan Connolly, Jim Cunningham, Ron Daniel, Jr., Jim Davis, Keith Dawson, Mark Day, Martin Duerst, David Durand, Lee Farrell, Chuck Fay, Roy Fielding, Mark Fisher, Alan Freier, George Florentine, Jim Gettys, Phill Hallam-Baker, Dennis Hamilton, Steve Henning, Alex Hopmann, Andre van der Hoek, Ben Laurie, Paul Leach, Ora Lassila, Karen MacArthur, Steven Martin,

Larry Masinter, Michael Mealling, Keith Moore, Henrik Nielsen,

Kenji Ota, Bob Parker, Glenn Peterson, Jon Radoff, Saveen Reddy, Henry Sanders, Christopher Seiwald, Judith Slein, Mike Spreitzer, Einar Stefferud, Ralph Swick, Kenji Takahashi, Robert Thau, John Turner, Sankar Virdhagriswaran, Fabio Vitali, Gregory Woodhouse, Lauren Wood

11 References

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

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

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

[Connolly et al, 1997] D. Connolly, R. Khare, H.F. Nielsen, "PEP - an Extension Mechanism for HTTP", Internet draft, work-in-progress. draft-ietf-http-pep-04.txt, ftp://ds.internic.net/internet-drafts/draft-ietf-http-pep-04.txt.

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

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

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

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

[Miller et al., 1996] J. Miller, T. Krauskopf, P. Resnick, W. Treese, "PICS Label Distribution Label Syntax and Communication

Protocols" Version 1.1, W3C Recommendation REC-PICS-labels-961031. http://www.w3.org/pub/WWW/TR/REC-PICS-labels-961031.html.

[Slein et al., 1997] J. A. Slein, F. Vitali, E. J. Whitehead, Jr., D. Durand, "Requirements for Distributed Authoring and Versioning on the World Wide Web." Internet-draft, work-in-progress, draft-ietf-webdav-requirements-04.txt, ftp://ds.internic.net/internet-drafts/draft-ietf-webdav-requirements-04.txt.

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

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

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

12 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