WEBDAV Working Group
INTERNET-DRAFT
<draft-ietf-webdav-protocol-00>

Y. Goland, Microsoft
E. J. Whitehead, Jr., U.C. Irvine
Asad Faizi, Netscape
Stephen R. Carter, Novell
Del Jensen, Novell

Expires January 15, 1997

July 13, 1997

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

Status of this Memo

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

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

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

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

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

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 Also known as small-chunk metadata, a hierarchical set of name/value pairs that describe 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.

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, 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 fully 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/attribute triples. The name of a property identifies the property s syntax and semantics, and provides an address with which to refer to a property. The value of a property is an octet stream. The attributes of a property are a set of name/value pairs that are not directly addressable. Attributes are retrieved in conjunction with retrieving a property, and are set when changing a property s value. This specification defines two attributes, live, which indicates if the property s syntax and semantics is enforced by the server, and readonly, which indicates that the property s value may be retrieved but not set.

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 and to be able to directly address that value.

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 certainly ubiquitous.

Many databases use a fixed schema mechanism, which 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 should not restrict RDBS from taking full advantage of their search facilities.

However, it seems that future trends might be toward hierarchical properties. As such, DAV requirements [] 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 for 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.

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 A but not a property A/B.

2.2.3 Property Attributes

The attributes of a property provide information about the property. Note that a property contains information about a resource.

Attributes consist of name/value pairs whose value MUST be a string. Attributes are not directly addressable, rather they are retrieved and defined in the context of other property operations. For example, if one retrieves a property value, the attributes will also be returned. If one sets a property value, one may also specify the values for its attributes.

All attributes on a server MUST be live. This means that the server MUST only record attributes with syntax and semantics the server understands and enforces. This normally means that clients can not define new attributes on a property; clients may only make use of the attributes supported by the server.

If a client submits an attribute when setting a property then the server MUST NOT record the property unless it accepts the values specified for the corresponding attributes. Thus, if a property value is submitted with a live attribute then the server MUST NOT record the value unless the server understands and enforces the syntax and semantics of the property.

2.2.4 Schemas

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

It is often useful to indicate support for a particular schema in a request or a response. Schema discovery is also useful for determining if a system supports a group of properties, attributes, 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.

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

properties and attributes that can be defined on a resource and

XML elements that can be returned in responses.

All DAV compliant servers MUST support the DAV schema.

2.3.1 Live Attribute

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

Purpose: To indicate that the property has its syntax and semantics enforced by the resource on which it is recorded.

Schema: http://www.ietf.org/standards/dav/

Parent: Any property

Values= = < >< >

Description: This attribute is used to indicate that the resource expressing the property understands and enforces

the syntax and semantics of the property. The absence of the Live attribute in a response indicates to the client that the corresponding property does not have its syntax and semantics enforced by the resource on which it is recorded. If a live attribute is included when setting the value of a property then the server SHOULD set the property if the property will be live and MUST NOT set the property if the property will not be live.

2.3.2 ReadOnly Attribute

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

Purpose: To indicate that a property can be retrieved, but not set through the property mechanism.

Schema: http://www.ietf.org/standards/dav/

Parent: Any property

Values= = < >< >

Description: This attribute is used to indicate that the property can only be retrieved, not set through the property mechanism. This attribute is not meant as an access control mechanism but rather to reflect the fact that the property is not designed to have its value set through the property mechanism. If this attribute is included when setting the value of a property, the request MUST be rejected since accepting the value would violate ReadOnly attribute. A server MUST NOT effect a property protocol element that is inconsistent or ill-defined with respect to the element s attribute state, were it to be expressed.

2.3.3 Elements

2.3.3.1 Prop XML element

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

Purpose: To specify the name and value of a property

Schema: http://www.ietf.org/standards/dav/

Parent: Any

Values: PropName PropValue

2.3.3.2 PropName XML element

Name: http://www.ietf.org/stnadards/dav/name

Purpose: To specify the name of a property, which MUST be a URI.

Schema: http://www.ietf.org/standards/dav/

Parent: Prop

Values: URI

2.3.3.3 PropValue XML element

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

Purpose: To specify the value of a property.

Schema: http://www.ietf.org/standards/dav/

Parent: Prop

Values: The contents of a property.

2.4 Property Identifiers

2.4.1 Problem Definition

The addition of DAV properties to the HTTP object model introduces the need for a mechanism to unambiguously refer to either the body of the resource or the properties of a resource.

2.4.2 Solution Requirement

The mechanism used for referring to the resource body must also be usable for referring to the resource s properties, such that even non-DAV aware clients can retrieve DAV properties.

2.4.3 DAV URL Parameter

To allow for the specification of property information in the context of an http scheme URL, a switch is needed. The switch indicates that following path segments specify a property location. To this end the DAV param is introduced for use with http scheme URLs. The path segment to the right of the DAV param MUST be formatted according to the XML Link standard, described in Appendix 3.

2.4.4 Name Encoding

Properties on a resource are given URIs as a name. Thus, in order to be able to refer to a property one must be able to put the property s URI into an HTTP URI.

For example, the author property with full name

http://www.w3.org/standards/z39.50/author is defined on http://somewhere.com/resource.

To create a reference to the author one would perform the following steps.

Add the DAV parameter to the base URI, http://somewhere.com/resource;DAV.

Add / to refer to the root of the resource s property namespace, http://somewhere.com/resource;DAV/.

Change the author property s name into parameter format by changing / s to ! s and encasing the entire value in parenthesis. The value must be encased in parenthesis in order to indicate the / to ! translation. The translation / to ! is done in order to prevent confusion over segments boundaries, and to make sure that the syntax for relative URIs remains well-defined. http://somewhere.com/resource;DAV/(http:!!www.w3.org!standards!z39.50!author).

The process is now complete, and the URL can be used in a GET or PATCH to retrieve or alter the value. See appendix 3 for more information.

2.4.5 Compatibility with legacy systems

2.4.5.1 Problem Definition

The HTTP parameter space is uncontrolled, thus someone may already be using a parameter with a value of DAV for some end other than the one described here. Thus a client sending a URI with a DAV param to a server may receive an unexpected or inappropriate response.

2.4.5.2 Solution Requirement

A mechanism is needed to prevent namespace collisions.

2.4.5.3 Proposed Solution

All DAV compliant servers MUST honor the DAV param type on http URLs. Thus if a client knows it is talking to a DAV

server, it can safely send an http URL with the DAV param.

The client may send the http URL with the DAV param extension to a server that is not known to be DAV compliant if the client uses PEP [Connolly, 1997] to prevent collisions. The proper PEP header is:

DAVPEP = PEP: {{map DAV }{strength must}}

Note: this PEP header is not compliant with [Connolly, 1997]; the PEP authors have indicated they will change the format to make the example legal.

2.5 Link XML Element

2.5.1 Problem Description

A mechanism is needed to associate resources with other resources. These associations, also 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.

2.5.2 Solution Requirements

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

2.5.3 Link XML Element

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

Purpose: The XML document which is the value of a link.

Schema: http://www.ietf.org/standards/dav/

Values= An XML document which MUST have a src and dst XML element.

Description: Link is used to provide the source and one or more destinations of the link. The type of the property provides the type of the link. Link is a multivalued element,so multiple Links may be used together to indicate multiple links with the same type.

2.5.4 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

2.5.5 Dst XML Element

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

Purpose: To indicate one or more destinations of a link

Schema: http://www.ietf.org/standards/dav/

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

Values= URI

2.5.6 Example

```
<XML>
     <Namespace><Ref>http://www.ietf.org/standards/dav/</><A
     S>D</></>
     <D:Prop>
          <Propname>Source</>
          <Propvalue>
               <XML:XML>
                    <Namespace>

                         <Ref>http://www.ietf.org/standards/
                    dav/</><AS>D</>
                    </>
                    <Namespace>

                         <Ref>http://www.foocorp.com/Project
                    /</><AS>F</>
                    </>
                    <D:Link>
                         <F:ProjectFiles>Source</>
                         <src>http://foo.bar/program</>

          <dst>http://foo.bar/source/main.c</>
```

```
                    </>
                    <D:Link>
                         <F:ProjectFiles>Library</>
                         <src>http://foo.bar/program</>

          <dst>http://foo.bar/source/main.lib</>
                    </>
                    <D:Link>
                         <F:ProjectFiles>Makefile</>
                         <src>http://foo.bar/program</>

          <dst>http://foo.bar/source/makefile</>
</>  </>  </>  </>  </>
```

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.

2.6 Properties and Methods

2.6.1 DELETE

The delete method, when used on a property, causes the property to be removed.

2.6.2 GET

A GET on a property returns the name of the 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 application/XML. If application/XML is used as the response format then it MUST include the http://www.ietf.org/standards/dav/ schema.

2.6.2.1 Example

```
GET bar;DAV/(http:!!www.w3.org!standards!z39.50!Authors)
HTTP/1.1
Host: foo.com

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

<XML>
     <XML:Namespace><Ref>http://www.ietf.org/standards/dav/<
/><AS>D</></>
     <XML:Namespace><Ref>http://www.w3.com/standards/z39.50/
</><AS>Z</></>
     <D:prop>
          <propname>Z:Authors</>
          <propvalue>
               <XML:XML>
                    <Namespace>

                         <Ref>http://www.ietf.org/standards/
                    dav/</>
                         <AS>D</>
                    </>
                    <Namespace>

                    <Ref>http://www.w3.com/standards/z39.50/
               </>
                         <AS>Z</>
                    </>
                    <Z:Author>Jane Doe</>
                    <Z:Author>Joe Doe</>
                    <Z:Author>Lots o Doe</>
</>  </>  </>  </>

GET bar;DAV/(Dublin:Producer) HTTP/1.1
Host: foo.com

HTTP/1.1 200 OK




Content-Type: application/xml
Content-Length: xxxx
E-tag:  2345
Last-Modified: xxxx
```

```
<XML>
     <XML:Namespace><Ref>http://www.ietf.org/standards/dav/<
/><AS>D</></>
     <XML:Namespace><Ref>Dublin</><AS>Du</></>
     <D:prop>
          <propname>Du:Producer</>
          <propvalue><XML:XML>Marcus Doe</></>
</>  </>

GET bar;DAV/ HTTP/1.1
Host: foo.com

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

<XML>
     <XML:Namespace><Ref>http://www.ietf.org/standards/dav/<
/><AS>D</></>
     <XML:Namespace><Ref>http://www.w3.com/standards/z39.50/
</><AS>Z</></>
     <XML:Namespace><Ref>Dublin</><AS>Du</></>
     <D:prop>
          <propname>Z:Authors</>
          <propvalue>
               <XML:XML>
                    <Namespace>

                         <Ref>http://www.ietf.org/standards/
                    dav/</>
                         <AS>D</>
                    </>
                    <Namespace>

                    <Ref>http://www.w3.com/standards/z39.50/
               </>
                         <AS>Z</>
                    </>
                    <Z:Author>Jane Doe</>
                    <Z:Author>Joe Doe</>
                    <Z:Author>Lots o Doe</>
     </>  </>  </>
     <D:prop>
          <propname>Du:Producer</>
          <propvalue><XML:XML>Marcus Doe</></>
</>  </>
```

2.6.3 PROPPATCH Method

The PROPPATCH method specifies how to alter a property. The message body controls the actual action taken by a PROPPATCH. All DAV compliant servers are required to support the use of the application/XML content-type using the http://www.ietf.org/standards/dav/proppatch/ schema in a PROPPATCH method with a request-URI that points to the resource upon which the property is defined.

The changes in a http://www.w3.com/standards/dav/proppatch/ request MUST be atomically executed, partial results are not allowed.

2.6.3.1 Request URI

The request URI of a PROPPATCH method with the http://www.ietf.org/standards/dav/proppatch/ schema MUST point to the resource upon which the property is defined.

2.6.3.2 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: <XML>

Values= *(Create | Remove | Insert)

Description: This XML element is a container for the information required to modify the properties on the resource. This XML element is multivalued.

2.6.3.3 Create XML element

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

Purpose: To create the DAV property 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 contains a Prop as the only element. The PropName contains the name of the property to be created or overwritten. The PropValue XML element contains the value of the new property.

2.6.3.4 Remove XML element

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

Purpose: To remove the DAV property specified inside the Remove XML element.

Schema: http://www.ietf.org/standards/dav/

Parent: http://www.ietf.org/standards/dav/PropertyUpdate

Values= PropName

Description: Remove specifies that the property specified in PropName should be removed. Specifying the removal of a property that does not exist is not an error.

2.6.3.5 Response Codes

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

400 Bad Request The client has provide a value whose syntax is illegal for the property.

401 Unauthorized The client does not have authorization to

alter one of the properties. This error also occurs if a property is inherently read only.

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.

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.

2.6.3.6 Example

```
PROPPATCH bar;DAV/ HTTP/1.1
Host: www.foo.com
Content-Type: application/XML
Content-Length: xxxx

<XML>
     <Namespace><Ref>http://www.ietf.org/standards/dav/</><A
S>D</></>
     <Namespace><Ref>http://www.w3.com/standards/z39.50/</><
AS>Z</></>
     <D:PropertyUpdate>
          <Create><prop>
               <propname>Z:authors</>
               <propvalue>
                    <XML:XML>
                    <Namespace>

     <Ref>http://www.ietf.org/standards/dav/proppatch/</>
                    <AS>D</>
                    </>
                    <Namespace>

     <Ref>http://www.w3.com/standards/z39.50/</>
                         <AS>Z</>
                    </>
                    <Z:Author>Jim Whitehead</>
                    <Z:Author>Roy Fielding</>
               </>
          </>
          <Remove><propname>Z:Copyright-Onwer</></>
</>  </>
```

2.6.4 PUT

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

2.6.5 SEARCH

2.6.5.1 Request-URI

The request-URI of the search method is the URI of the

resource. .

The Depth header MUST NOT be used on a SEARCH method which contains a Limited-Search XML element (limited search).

2.6.5.2 Command Format

The message body stipulates the action of a SEARCH method. This section defines an application/xml content type using the http://www.ietf.org/standards/dav/ schema. This method is not normally cacheable.

2.6.5.2.1 Limited-Search XML element

Name: http://www.ietf.org/standards/dav/limited-search

Purpose: To specify the set of matching properties

Schema: http://www.ietf.org/standards/dav/

Parent: <XML>

Values: The value is a single OR XML element. The OR element may only contain AND XML elements, and MUST contain at least one AND element.

Description: This property indicates a very limited search. The search may only be on HTTP properties.

2.6.5.2.2 OR XML element

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

Purpose: To take its members, evaluate them, get a true or false result, or the results together, and have that be the total result.

Schema: http://www.ietf.org/standards/dav/

Parent: Limited-Search XML element

Values: AND XML element.

2.6.5.2.3 AND XML element

Name: http://www.ietf.org/sandards/dav/and

Purpose: To take its members, evaluate them, get a true or false result, and the results together, and have that be the total result.

Schema: http://www.ietf.org/standards/dav

Parent: OR XML element

Values: Zero or one Name XML element, and zero or one Value XML element. There MUST be at least one Name or Value XML element.

2.6.5.2.4 Name XML element

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

Purpose: To provide a pattern against which property names are to be compared. If the name matches then the property evaluates to true, otherwise false.

Schema: http://www.ietf.org/standards/dav/

Parent: AND XML element

Values: Match-Stream

2.6.5.2.5 Value XML element

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

Purpose: To provide a pattern against which property values are to be compared. If the value matches then the property evaluates to true, otherwise false.

Schema: http://www.ietf.org/standards/dav/

Parent: AND XML element

Values: Match-Stream

2.6.5.2.6 Match-String XML element

Name: http://www.ietf.org/standards/dav/match-string

Purpose: To specify a search pattern to be matched against an octet stream

Schema: http://www.ietf.org/standards/dav/

Parent: Name or Value XML element

Values: (* | ? | EncodedOctet)
EncodedOctet = <An EncodedOctet uses XML encoding to encode * and ? as well as < and >

Description: This element provides a template against which anything that can be expressed as an octet stream may be compared. * is a wildcard that matches zero or more unspecified contiguous octets. ? is a wildcard that matches exactly one unspecified octet.

2.6.5.3 Response Format

The response is an application/xml message body which contains a single SearchResult XML element whose contents are a series of XML elements representing matching properties.

2.6.5.3.1 SearchResult XML element

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

Purpose: To contain the results of a SEARCH request

Schema: http://www.ietf.org/standards/dav/

Parent: Any, usually <XML>

Values: Zero or more Prop XML elements (defined in Properties draft)

Description: The SearchResult XML element provides the context to inform the client that its contents are not just some XML element, but an XML representation of the requested property.

2.6.5.4 Example

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

```
<XML>
     <XML:Namespace>
          <Ref>http://www.ietf.org/standards/dav/</>
          <AS>S</>
     </>
     <S:limited-search>




          <OR>
               <AND>
                    <Name>*</>
               </>
          </>
     </>
</>

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

<XML>
     <XML:Namespace>
          <Ref>http://www.ietf.org/standards/dav/</>
          <As>S</>
     </>
     <XML:Namespace>
          <Ref>http://www.foo.bar/boxschema</>
          <AS>R</>
     </>
     <S:SearchResult>
          <Prop>
               <PropName>R:bigbox</>
               <PropValue>
                    <XML:XML>
                         <BoxType>Box type A</>
                    </>
               </>
          </>
          <Prop>
               <PropName>R:author</>
               <PropValue>
                    <XML:XML>
                         <Name>J.J.
Dingleheimerschmidt</>
                    </>
               </>
```

```
            </>
        </>
    </>
```

The result will return all properties on the container and its members. In this case only two properties were found, one on the container and one on one of its members, both properties are live.

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

3.1 Observations on the HTTP Object Model

As a prerequisite for specification of collections and name space operations for the Web, a model for collection resources and for namespace topology must be given. This section describes a new type of Web resource, the collection resource, and provides a model for discussing the relationship between the resources that are generated as the result of a data-producing process, and the source resources that describe the process.

3.1.1 Collection Resources

A collection is a Web resource type whose primary state is a set of URIs and associated values that are recorded as properties on the resource. The URIs identify resources that are members of the collection. The values associated with each URI include information such as the Last Modified Date, Entity Tag, Creation Date, Content Type, Display Name, and whether the member is a collection.

A member of a collection is either an internal member resource, which MUST have a URI that is relative to the base URI of the collection, or an external member resource, which has a URI which is not relative to the base URI of the collection. External member resources are further subdivided into propagate members, which have recursive method invocations propagated to them, and no-propagate members, which do not.

A collection resource may be viewed and used as a compound resource in which the collection is a container for a group of related resources that, together, form a larger logical unit. For example, a collection of HTML resources where

each resource is the chapter of a book can be viewed as a compound resource representing the entire book.

Some methods, when invoked on a collection, affect the entire collection. For example, it is possible to copy an entire collection and its contents with just a single copy method request. The model for performing these operations is a tree traversal. The method is invoked on the collection, which then performs the method on itself before propagating the method to all its internal members and propagate external members. If these are non-collection resources, the request method is processed. However, if the request is propagated to another collection, then the propagation begins again. This sequence of actions causes the method to be propagated as a tree traversal of the members of the collections. It is incumbent upon the client to perform any locking operation on the collection or subordinate members that it deems necessary in order to maintain state consistency during the execution of such methods.

3.1.2 Creation and Retrieval of Collection Resources

Since the existing HTTP methods for creating (PUT, POST) and retrieving (GET) a resource were defined for non-collection resources, it is not surprising that the semantics of these methods do not transfer well to collections. For example, the PUT method is defined to store the request entity under the 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 provide separate access control for collection creation and other uses of POST if they both use the same method.

The GET method when applied to collections is also

problematic. 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.

Because of the difficulty of reusing some existing HTTP/1.1 methods for collections, two new resource creation/retrieval methods are needed. This specification introduces the MKCOL method for creating collection resources, and the INDEX method for retrieving the contents of a collection.

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

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, HTML files can sometimes be processed by the server before being transmitted as a return entity body. 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 file 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, the fact that the source resource(s) for server generated output do not have a mapping to the URI namespace is not a problem, and has desirable security benefits. However, if remote editing of the source resource(s) is desired, they 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 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.

In the general case, a large number of source resources can comprise a data-producing process that generates many output resources, creating a many-to-many relationship between output resources and source resources. If each output resource had links back to every source resource in the data-producing process, there can be a potentially large number of such links. Due to the potentially large number of links, and the lack of a policy for ordering access to multiple sources, explicit storage of source relationships is limited to cases with only a single source resource.

3.2 MKCOL Method

3.2.1 Problem Description

The client needs a way 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 a failure to the principal.

Requires that the server MAY, if necessary, create any intermediate collections so that the underlying storage

medium is self-consistent.

3.2.3 Request

The MKCOL method 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 MAY add the Request-URI to one or more collections within the server s controlled namespace.

3.2.3.1 MKCOL Without Request Body

When MKCOL is invoked without a request body then the collection created 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.

3.2.3.3 Creating Multiple Collections

The server MAY create intermediate collections if they do not already exist. For example, if the collection http://server/a/ already exists in the server s namespace, then while performing a MKCOL to create http://server/a/b/c/ the server may also create a collection at http://server/a/b/.

3.2.4 Response

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

201 (Created) - The structured resource was created in its entirety.

403 (Forbidden) - The server does not allow the creation of collections at the given location in its namespace.

415 (Unsupported Media Type) The server does not support the request type of the 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.

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 INDEX Method

3.3.1 Problem Description

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

3.3.2 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

3.3.3 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 machine-readable list of its members. For other resources, the information returned by INDEX is undefined, and MAY vary. The request message body of an INDEX request SHOULD be ignored.

The Depth header can be used to indicate how much of a result can be generated for the response. The specific values allowed for the depth header when used with the INDEX

method are 1 and infinity. The 1 value indicates that the internal and external member resources should be reported in the result, infinity indicates that all internal and external member resources and all their descendants should be in the result. If the Depth header is not given, then 1 is assumed. Servers MUST honor a depth of 1. Servers MAY honor infinity. If the server does not support the value of the depth header then a 412 (Precondition failed) MUST be returned.

3.3.4 The Response

200 (OK) The server MUST send an application/xml response entity which describes the collection.

404 (Not Found) - Same behavior as HTTP 1.1. The server never had the resource, or the server permanently deleted the resource and has no knowledge that it ever existed. This error code implies that, essentially, the server has no information about the Request URI.

3.3.5 Response Message Body

The default INDEX response for a resource is an application/xml HTTP entity (i.e., an Extensible Markup Language (XML) document) that contains a single XML element called collectionresource which describes the collection, and a set of XML elements called memberesource which describe the members of the collection.

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.

The server MUST transmit the following XML elements for each member resource of a collection: Ref, IsCollection, Content-Type, External. The server MUST transmit the following XML elements if it can generate any meaningful values for them: Creation-Date, Last-Modified, DisplayName, Content-Language.
The server SHOULD transmit Etag XML elements for each member (see section 13.3.4 of RFC 2068).

The value of content-type, last-modified, and etag XML elements MUST be identical to the value of the response header field of the same name in the HTTP/1.1 specification.

Since the HTTP/1.1 header fields are described in terms of the on-the-wire entity, the values presented by INDEX are those that would be generated if the resource was accessed using the GET method without content negotiation.

3.3.5.1 CollectionResource

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

Purpose: Describes a collection

Schema: http://www.ietf.org/standards/dav/

Parent: <XML>

Value:MemberResource

3.3.5.2 MemberResource

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

Purpose: Describes a member of a collection

Schema: http://www.ietf.org/standards/dav/

Parent: CollectionResource

Value:Ref, IsCollection, Content-Type, External, Creation-Date, Last-Modified, ETag, DisplayName (other XML elements MAY also be present)

3.3.5.3 Ref

See XML definition.

3.3.5.4 IsCollection

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

Purpose: This is a boolean value which is set to true if the entry is a collection

Schema: http://www.ietf.org/standards/dav/

Parent: MemberResource

Value:(true | false)

3.3.5.5 Content-Type

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: MemberResource

Value:media-type ; defined in Section 3.7 of [HTTP11]
If no meaningful content-type can be generated, then an empty value MUST be given.

3.3.5.6 External

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

Purpose: If present, this element indicates the resource is an external member of the collection. If the value is propagate, it is a propagate member, if the value is no-propagate, it is a no-propagate member.

Schema: http://www.ietf.org/standards/dav/

Parent: MemberResource

Value:(propagate | no-propagate)

3.3.5.7 Creation-Date

Name: http://www.ietf.org/standards/dav/creation-date

Purpose: The date the resource was created.

Schema: http://www.ietf.org/standards/dav/

Parent: MemberResource

Value:The date MUST be given in RFC 1123 format (rfc-1123 production, defined in section 3.3.1 of [HTTP11]

3.3.5.8 Last-Modified

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: MemberResource

Value:The date MUST be given in RFC 1123 format (rfc-1123 production, defined in section 3.3.1 of [HTTP11]

3.3.5.9 ETag

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

Purpose: The entity tag of the resource.

Schema: http://www.ietf.org/standards/dav/

Parent: MemberResource

Value:entity-tag   ; defined in Section 3.11 of [HTTP11]

3.3.5.10 DisplayName

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

Purpose: A name for the resource that is suitable for presentation to a person

Schema: http://www.ietf.org/standards/dav/

Parent: MemberResource

Value:Any valid XML character data (from XML specification)

3.3.5.11 Content-Language

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

Purpose: Describes the natural language(s) of the intended audience for the resource.

Schema: http://www.ietf.org.standards/dav/

Parent: MemberResource

Value: 1#language-tag ;language-tag is defined in section 14.13 of RFC 2068

3.3.6 Example

```
INDEX /user/yarong/dav_drafts/ HTTP/1.1
Host: www.microsoft.com
Depth: 1

HTTP/1.1 200 OK
Content-Type: application/xml
Content-Length: xxx
Last-Modified: xxx
ETag:  fooyyybar


<XML>
<XML:Namespace><Ref>http://www.ietf.org/standards/dav/<
/><As>D</></>
<D:CollectionResource>
     <MemberResource>
          <XML:Ref>namespace.doc</>
          <IsCollection>false</>
          <Content-Type>application/msword</>
          <External>false</>
          <Creation-Date>Thu, 20 Mar 1997 23:05:25
GMT</>
          <Last-Modified>Fri, 22 Aug 1997 18:22:56
GMT</>
          <Etag>8675309</>
          <DisplayName>WebDAV Name Space Operations
Draft</>
          <Content-Language>en</>
</>  </>
</>
```

This example shows the result of the INDEX method applied to the collection resource http://www.microsoft.com/er/yarong/dav_drafts/. It returns a response body in XML format, which gives information about the container s sole member, http://www.microsoft.com/users/yarong/dav_drafts/namespace.doc.

3.4 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.

3.4.1 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 [HTTP11]. 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 do not require any modification when applied to collection resources.

3.4.2 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 modified because it is largely undefined. Thus the semantics of POST do not require any modification when applied to a collection.

3.4.3 PUT for Collections

In HTTP/1.1, PUT stores the request entity under the Request-URI, and hence its semantics are limited to non-collection resources. If a PUT is invoked on a collection resource it MUST fail.

When the PUT operation creates a new non-collection resource, a server MAY add that resource s URI to one or more collections within the server s controlled namespace.

3.4.4 DELETE for Collections

When DELETE is applied to a collection resource, all internal members MUST be recursively deleted. The dav:link/propagate external members MUST be deleted and their links must be removed. dav:link/nopropagate external members MUST have only their link removed; the resources MUST not be deleted.

The Depth header does not apply to the DELETE method. It cannot be used to limit the extent of the operation. If it

is present it MUST be ignored.

When applied to any resource, the DELETE method deletes all properties on the Request-URI.

During DELETE processing, a server MAY remove the URI of the deleted resource(s) from collections within its controlled namespace.

3.4.4.1 New Response Codes for DELETE

207 (Partial Success) Only some of the member resources were deleted. The response entity will describe any errors.

500 (Server Error) The resource was in such a state that it could not be deleted. The response entity will describe reason for the error.

3.5 COPY Method

3.5.1 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.

3.5.2 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.

3.5.3 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.

3.5.3.1 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 .

If the Duplicate-Properties header is false, then properties SHOULD NOT be copied to the destination resource. If the Duplicate-Properties header is false and the Enforce-Live-Properties header is also present, the request MUST fail with a 412 (Precondition Failed) status code. [Ed. Note: what if resource to be copied has no properties, and no properties are explicitly named in the header?]

All properties on a source resource SHOULD be duplicated on the destination resource following the definition for copying properties.

3.5.3.2 COPY for Properties

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 in an identically named, dead resource on the destination resource.

For every dead property defined on the source resource, there SHOULD be an octet-for-octet identical dead property on the destination resource.

3.5.3.3 COPY for Collections

The Depth and Overwrite headers govern the behavior of COPY for collections.

When performing a recursive copy, all HTTP/1.1 request headers are duplicated on the propagated method request except for the precondition headers If-Modified-Since, If-Match, If-None-Match, If-Range, If-Unmodified-Since, which should only be applied to the Request-URI in order to determine if the operation should be performed. The Destination header MUST be rewritten to preserve the membership of the destination collection, i.e., by appending the relative URI of the member to the URI of the destination collection.

A Depth of 0 indicates the collection MUST duplicate all of its external members in a new collection at the Destination. Since the COPY method is not propagated to its members, no internal member resource is duplicated.

A Depth of 1 indicates the collection MUST propagate the COPY to all internal, non-collection members. If the Overwrite header is true the COPY method duplicates all of its external members in a new collection at the Destination. If the Overwrite header is false and the destination resource is a collection, the COPY method does not duplicate its external members, but is propagated to all internal, non-collection members.

A Depth of infinity indicates the collection MUST propagate the COPY method to all internal members. If the Overwrite header is true, the COPY method MUST duplicate all of its external members in a new collection at the Destination. If the Overwrite header is false and the destination resource is a collection, then the COPY method does not duplicate its external members, but is propagated to all internal members.

3.5.3.4 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.

3.5.4 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.

207 (Partial Success) Only some of the member resources were copied. The return entity body describes the status code for each 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.

500 (Server Error) The resource was in such a state that it could not be copied. This may occur if the Destination header indicated an external (from the point of view of the server) resource and the server has no capability to copy to an external resource.

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

3.5.5 Examples

3.5.5.1 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
Overwrite:  true


HTTP/1.1 200 OK
```

3.5.5.2 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


HTTP/1.1 412 Precondition Failed
```

3.6 MOVE Method

3.6.1 Problem Description

The move operation on a resource is the logical equivalent of a copy followed by a delete.

In HTTP 1.1, the 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 DAV 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.

3.6.2 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.

3.6.3 The Request

The MOVE method is defined as the logical equivalent of a COPY followed by a DELETE of the source resource, performed atomically.

3.6.4 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.

207 (Partial Success) Only some of the member resources were moved. The return entity body will give the status code for each 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.

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.

3.6.5 Examples

3.6.5.1 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
Overwrite: true
```

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

3.7 Multi-Status Response

3.7.1 Problem Definition

Certain methods (COPY, MOVE, and DELETE) when applied to a collection might be recursively applied to all sub-members of the collection. In this case, it is possible that the operation will succeed on some member resources and fail on others, thus generating a 207 (Partial Success) status code. A principal may need to know which members of the collection succeeded and which failed.

3.7.2 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

3.7.3 Multi-Status Response

The default multi-status response body is an application/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. Each response XML element contains two sub-entities, ref, the URI of the resource on which the method was invoked, and status, which holds the status-line from the method invocation.

A multi-status response MUST be present when a 207 (Partial Success) status code is returned by the initial method invocation.

3.7.3.1 MultiResponse

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

Purpose: Contains multiple response messages.

Schema: http://www.ietf.org/standards/dav/multiresponse/

Parent: <XML>

Value:1*Response

3.7.3.2 Response

Name: http://www.ietf.org/standards/dav/multiresponse/response

Purpose: Holds a single response

Schema: http://www.ietf.org/standards/dav/multiresponse/

Parent: MultiResponse

Value:Ref, Status

3.7.3.3 Status

Name: http://www.ietf.org/standards/dav/multiresponse/status

Purpose: Holds a single HTTP status-line

Schema: http://www.ietf.org/standards/dav/multiresponse/

Parent: Response

Value:status-line ;status-line defined in [HTTP11]

3.7.4 Example

```
COPY /users/jdoe/collection/ HTTP/1.1
Host: www.doecorp.com
Destination:
http://www.doecorp.com/users/jdoe/othercollection/
Depth: infinity
Overwrite: false


HTTP/1.1 207 Partial Success
Content-Type: application/xml
Content-Length: xxx

<XML>
<XML:Namespace><Ref>http://www.ietf.org/standards/dav/multir
esponse/</><As>R</></>
<R:MultiResponse>
     <Response>

     <XML:Ref>http://www.doecorp.com/users/jdoe/collection/i
ndex.html</>
          <Status>HTTP/1.1 412 Precondition Failed</>
     </>
     <Response>
```

```
     <XML:Ref>http://www.doecorp.com/users/jdoe/collection/r
eport.html</>
          <Status>HTTP/1.1 200 OK</>
     </>
</>
</>
```

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. The propagation type of the external URI is specified in the Collection-Member Header.

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.

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 PATCH method contains a list of differences between the original version of 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.

Since the semantics of PATCH are non-idempotent, responses to this method are not cacheable.

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.

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

3.10.4 application/XML elements for PATCH

The resourceupdate XML elementXML 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).

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: <XML>

Value:*(Insert | Delete | Replace)

3.10.4.2 Insert

Name: http://www.ietf.org/standards/dav/patch/insert

Purpose: Insert the XML elementXML element s contents starting exactly at the specified octet.

Schema: http://www.ietf.org/standards/dav/patch/

Parent: ResourceUpdate

Value:The insert XML elementXML element MUST contain an octet XML elementXML 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 elementXML element contains the octets to be inserted.

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 elementXML element MUST contain an octet-range XML elementXML element. The value of this XML elementXML element is empty.

Discussion: The octets which 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.

3.10.4.5 Octet-Range Attribute

Name:
http://www.ietf.org/standards/dav/patch/octet-range

Purpose: Specifies a range of octets which 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 but 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. Octet values MUST specify locations in the state of the resource prior to the processing of the PATCH method.

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.

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:

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

<XML>
<XML:Namespace><ref>http://www.ietf.org/standards/dav/patch/</><AS>D</></>
<D:ResourceUpdate>
     <Replace><octet-range>14</>&003CTITLE&003ENew Title&003C/TITLE&003E</>
     <Delete><octet-range>38-50</>
     <Insert><octet-range>86</>&003CP&003ENew paragraph&003C/P&003E
</>
</></>
```

Response:

```
HTTP/1.1 100 Continue

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 Depth

The Depth header determines the depth to which a method is propagated on a resource s children.

```
Depth =  Depth   :  DepthToken
DepthToken = "0" | "1" | "infinity" | token
```

The optional token allows for extension. A server MUST ignore a Depth header with an unknown value.

3.11.2 Destination

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

```
Destination=  Destination   :  URI
```

3.11.3 Enforce-Live-Properties

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.

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

3.11.4 Duplicate-Properties

The Duplicate-Properties header instructs the server whether to duplicate the source resource s properties onto the

destination resource during a COPY or MOVE. A value of false requires that the server MUST NOT duplicate on the destination resource any properties that are defined on the source resource. By default, the value of this header is true, and a client MAY omit this header from a request when its value is true.

```
Duplicate-Properties = Duplicate-Properties :
( true | false )
```

3.11.5 Overwrite

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 false, and a client MAY omit this header from a request when its value is false. 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 )
```

3.11.6 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 PEP, allows the client to specify the end result they desire. The Destroy header is specified as follows:

```
DestroyHeader = "Destroy" ":" #Choices
Choices = "VersionDestroy" | "NoUndelete" | "Undelete"
| Token
```

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.

3.11.7 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   :  PropType SP URI
PropType =  propagation   =  ( prop  |  noprop )
```

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, DAV asserts no policy on partial 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 extensible to 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 Problem Description - Overview

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

This draft allows locks to vary over two parameters, the number of principals involved and the type of access to be granted. This draft will only provide for the definition of locking for one access type, write. However, the syntax is extensible enough to allow for the specification of other access types. It is a goal of this proposal that it use the same access verbs as will be defined in the access control draft.

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 (probably) 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 in conjunction with a precondition header (If-State-Match) that checks for existence of the lock prior to writing the resource. 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). The intent of a shared lock is to let collaborators know who else is potentially working on a resource..

Why not use exclusive write locks all the time? Experience from initial 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. Further, what if the administrator can't be reached immediately?

Despite their potential problems, exclusive write locks are extremely useful, since often a guarantee of freedom from overwrite conflicts is exactly what is needed. The solution: provide exclusive write locks, but also 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 DAV 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. 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 DAV.

5.2 LOCK Method

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 a Lock-Token header. If the server supports a time based lock removal mechanism on the resource, a successful lock invocation SHOULD return a Time-Out header.

5.2.2 The Effect of Locks on Properties and Containers

By default a lock affects the entire state of the resource, including its associated properties. As such it is illegal to specify a lock on a property. 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. The Depth header expresses the general semantics of a LOCK method request when invoked on a collection (note that specific lock types may restrict the effect of a lock, for example limiting the allowable values of the Depth header):

A Depth header (defined in the namespace draft) may be used

on a LOCK method when the LOCK method is applied to a collection resource. The legal values for Depth on a LOCK are 0, 1, and Infinity. A Depth of 0 instructs the resource to just lock the container. As previously mentioned, depending on the type of lock, the lock affects the ability to add or remove members of the container.

@.A Depth of 1 means that the container is locked and a LOCK is executed on the container s propagate members with a Depth of 0 and If-Range, If-Modified-Since, If-Unmodified-Since, If-Match and If-None-Match headers are dropped. However, the effects of the LOCK MUST be atomic in that either the container and all of its members are locked or no lock is granted. The result of a Depth 1 lock is a single lock token which represents the lock on the container and all of its members. This lock token may be used in an If-State-Match or If-Not-State-Match header against any of the resources covered by the lock. Since the lock token represents a lock on all the resources, an UNLOCK using that token will remove the lock from all included resources, not just the resource the UNLOCK was executed on.

@.A Depth of infinity means that the LOCK is recursively executed, with a Depth of infinity, on the collection and all of its propagate members and all of their propagate members. As with a Depth of 1, the LOCK must be granted in total or not at all. Otherwise the lock operates in the same manner as a Depth of 1 lock.

The default behavior when locking a container is to act as if a Depth: 0 header had been placed on the method.

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 Interaction with other Methods

Only two methods, MOVE and DELETE, have side effects which involve locks. When a resource is moved, its lock SHOULD be moved with it. However this may not always be possible and there is currently no proposal to create a header which would specify that the lock request should fail if the resource s locks can not be maintained. A COPY MUST NOT copy any locks on the source resource over to the destination

resource. Deleting a resource MUST remove all locks on the resource.

5.2.5 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 lock is re-requested by the principal who owns the lock, the lock MAY be regranted. If the lock is regranted, the same lock token that was previously issued MUST be returned.

5.2.6 Status Codes

412 Precondition Failed The included state-token was not enforceable on this resource.

416 Locked The resource is locked so the method has been rejected.

5.2.7 Example

```
LOCK /workspace/webdav/proposal.doc HTTP/1.1
Host: webdav.sb.aol.com
Lock-Info: LockType=Write LockScope=Exclusive
Owner: <http://www.ics.uci.edu/~ejw/contact.html>
```

```
HTTP/1.1 200 OK
State-Token: StateToken:Type=^DAV:/LOCK/DAVLOCK^:Res=^http:/
/www.ics.uci.edu/workspace/webdav/proposal.doc^:LockType=Wri
te:LockScope=Exclusive:ServerID=12382349AdfFFF
Time-Out: ClockType=Activity TimeType=second;604800
```

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.

5.2.8 Lock-Info Request Header

The Lock-Info 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 CRLF
DAVLockType =  LockType   =  DAVLockTypeValue
DAVLockTypeValue = ( Write  | *(uchar | reserved))
DAVLockScope =  LockScope   =  DAVLockScopeValue
DAVLockScopeValue = ( Exclusive  | Shared  | *(uchar |
reserved))
```

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.

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 human. 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 human.

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 field may or may not support including the owners name and that name is often set to an alias anyway. Therefore a header more flexible than From is required.

5.2.9.3 Syntax

Owner = "Owner" ":" ((< URI >) | quoted-string)

The URI SHOULD provide a means 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). The quoted string SHOULD provide a means for directly contacting the principal, 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, use the resource as needed, and then remove the lock when it is no longer needed. However, this scenario is frequently not completed, leaving active but unused locks. Reasons for this include client programs crashing and loosing 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 remove. In the first case a principal other than the lock owner has sufficient access rights to order the lock removed, even though they did not take it out. User-agents MUST assume that such a mechanism is available and thus 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.

The second solution, is the time based removal policy. Activity based systems set a timer as soon as the lock is taken out. Every time a method is executed on the resource, the timer is reset. If the timer runs out, the lock is removed.

Finally, some systems only allow locks to exist for the duration of a session, where a session is defined as the time when the HTTP connection that was used to take out the lock remains connected. This mechanism is used to allow programs which are likely to be improperly exited, such as JAVA programs running in a browser, to take out locks without leaving a lot of ownerless locks around when they are improperly exited.

5.2.10.3 Syntax

```
TimeOut = "Time-Out" ":" ((TimeOutType SP Session) |
TimeOutVal |
          Session) CRLF
TimeOutType = ClockType SP TimeType
ClockType =  ClockType   =  ClockTypeValue
ClockTypeValue =  Activity
TimeType =  TimeType   =  TimeTypeValue
TimeTypeValue =  Second   ;  DAVTimeOutVal
DAVTimeOutVal = 1*digit
Session =  Session   =  ( Yes  |  No )
```

The Second TimeType specifies the number of seconds that may elapse before the lock is automatically removed. A server MUST not generate a time out value for second greater than 2^32-1.

If no time based system is in use then a Time-Out header MUST NOT be returned. The Time-Out header MUST only be returned in a response to a LOCK request.When session is set to yes then whatever clocktype and timetype is being used,

their effects are scoped within that particular session. So an absolute lock with a ten day expiration period will only remain active so long as the session remains active. A DAVTimeOutVal value must be greater than zero.

Clients MAY include TimeOut headers in their LOCK requests. However the server is not required to honor or even consider the request. The primary purpose in allowing clients to submit a TimeOut header is to inform the server if the client is requesting a session based lock. If a timeout is associated with the lock, the server MUST return a TimeOut header with a valid value.

5.2.11 State-Token Header

5.2.11.1 Problem Definition

Program A, used by User A, takes out a write lock on a resource. Program B, also run by User A, then proceeds to perform a PUT to the locked resource. The PUT will succeed because locks are associated with a principal, not a program, and thus program B, because it is acting with principal A s credential, will be allowed to perform the PUT. In reality program B had no knowledge of the lock and had it had such knowledge, would not have overwritten the resource. Hence, a mechanism is needed to prevent different programs from accidentally ignoring locks taken out by other programs with the same authorization.

5.2.11.2 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.

5.2.11.3 State-Token Header

The State-Token header is returned in a successful response to the LOCK method or is used as a request header with the UNLOCK method.

The State-Token header containing a lock token owned by the request principal is used by the principal on arbitrary method to indicate that the principal is aware of the specified lock. If the State-Token header with the appropriate lock token is not included the request MUST be rejected, even though the requesting principal has authorization to make modifications specified by the lock type. This injunction does not apply to methods that are not affected by the principal s lock.

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.

Note that 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 Write Lock

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

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 locked.

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.

Write locking a container also prevents adding or removing members of the container. This means that attempts to PUT/POST a resource into the immediate name space of the write locked container MUST fail if the principal requesting the action does not have the write lock on the container. In order to keep the behavior of locking containers consistent all locks on containers MUST contain a Depth header equal to infinity, any other value is illegal.

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 in effect. Thus 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 Proposed Solution

The proposed solution is to provide a lock token in the response of a lock request. The lock token is a type of state token and describes a particular lock. The same lock token must never be repeated on a particular resource. This prevents problems with long held outstanding lock tokens being confused with newer tokens. This uniqueness requirement is the same as for e-tags. This requirement also allows for tokens to be submitted across resources and servers without fear of confusion.

5.4.3 Lock Token Definition

The lock token is returned in the State-Token header in the response to a LOCK method. The lock token can also be discovered through lock discovery on a resource.

Lock-Token-URL = StateToken: Type : Resources : State-

Info
Type = Type = ^DAV:/LOCK/DAVLOCK^

Resources = Res = 1*(^ Caret-Encoded-URI ^)
Caret-Encoded-URI = <This is a URI which has all ^ s %
encoded.>
State-Info = DAVLockScope : DAVLockType : ServerID ;
DAVLockScope, DAVLockType defined in Lock-Info header
ServerID = ServerID = *(uchar | reserved)

The ServerID is a field for use by the server. Its most basic purpose is to put in a unique identifier to guarantee that a server will never confuse an old lock token with a newer one. However the server is free to use the field to record whatever information it deems fit. The field is opaque to clients.

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.

5.5.2 Example

UNLOCK /workspace/webdav/proposal.doc HTTP/1.1
Host: webdav.sb.aol.com
State-Token: StateToken:Type=^DAV:/LOCK/DAVLOCK^:Res=^http:/
/www.ics.uci.edu/workspace/webdav/proposal.doc^:LockType=Wri
te:LockScope=Exclusive:ServerID=12382349AdfFFF

HTTP/1.1 200 OK

In this example, the lock from example of Section 2.9 is removed from the resource at
http://webdav.sb.aol.com/workspace/webdav/proposal.doc

5.6 Discovery Mechanisms

5.6.1 Lock Type 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 can perform some form of discovery to determine what lock types the server actually supports, then formulate a supported request. This is known as lock type discovery. Lock type discovery is not the same as discovering what access control types are supported, as 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 types 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: HYPERLINK 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

5.6.2 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 in particular 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.

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 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= CLOCKTYPE TIMETYPE TIMEOUTVAL

5.6.2.7 CLOCKTYPE XML Element

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

Purpose: Returns the clock type used with this lock

Schema: http://www.ietf.org/standards/dav/

Parent: TIMEOUT

Values= ClockTypeValue

5.6.2.8 TIMETYPE XML Element

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

Purpose: Returns the time type used with this lock

Schema: http://www.ietf.org/standards/dav/

Parent: TIMEOUT

Values= TimeTypeValue

5.6.2.9 TIMEOUTVAL XML Element

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

Purpose: Returns the amount of time left on the lock

Schema: http://www.ietf.org/standards/dav/

Parent: TIMEOUT

Values= DAVTimeOutVal

5.6.2.10 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= XML:REF

Description: The REF contains a Lock-Token-URL.

6 Version Control

[TBD]

7 Internationalization Support

[TBD]

8 Security Considerations

[TBD]

9 Acknowledgements

Roy Fielding, Richard Taylor, Larry Masinter, Henry Sanders, Judith Slein, Dan Connolly, David Durand, Henrik Nielsen, Paul Leach. Kenji Ota, Kenji Takahashi. Jim Cunningham. Others, TBD.

10 References

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

[Bray, 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, 1997] D. Connolly, R. Khare, H.F. Nielsen, PEP - an Extension Mechanism for HTTP , Internet draft, work-in-progress. draft-ietf-http-pep-03.txt, ftp://ds.internic.net/internet-drafts/draft-ietf-http-pep-03.txt.

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

[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.

[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.

11 Authors' Addresses

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

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

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

Stephen R Carter
Novell
1555 N. Technology Way
M/S ORM F111
Orem, UT 84097-2399
Fax (801) 228 5176
Email srcarter@novell.com

Del Jensen
Novell
1555 N. Technology Way
M/S ORM F111
Orem, UT 84097-2399
Fax (801) 228 5176
Email dcjensen@novell.com

Appendix 1 - Content Type Application/XML

This is a digest of the XML data specification available at http://www.w3.org/TR/WD-xml-lang.html

Syntax

The application/XML content type contains an XML document. Its syntax is as defined below.

```
XML = XMLStart *XMLEntity Close

XMLStart =  <   XML   >

XMLEntity= Open *(XMLText | XMLEntity) Close

Close =  </>  |  <  / Entity-Name Markup >

Open =  <  Entity-Name *Attribute  >

Attribute = Entity-Name  =  Quoted-String

XMLText = <An Octet Stream which uses XML encoding for  <
and  > >

Entity-Name = ([As-Tag  : ] Name) | (As-Tag  : )
```

As-Tag = 1*Alpha

Name = (alpha | _) *(alpha | digit | . | - | _ | other)

Other = <Other characters must be encoded>

XML element
An XML element, as defined in the BNF, is an open tag with content followed by a close tag. In order to prevent confusion with the term entity as used in HTTP, the term XML element will be used.

The first XML element of a XML document MUST be the <XML> XML element. This XML element tells the parser that it is dealing with an XML document. So long as this XML element is present the parser can be sure that it can parse the document, even if XML has been extended. If XML is ever altered in a manner that is not backwards compatible with this specification then the content-type and the outer most XML element MUST be changed.

Entity-Name
All XML element names must map to URIs. However due to historical restrictions on what characters may appear in an XML element name, URIs cannot be expressed in an XML element name. This issue is dealt with in more detail in section 10.

Entity-Names without [AS] are relative URIs whose base is the enclosing Entity-Name. If the enclosing Entity-Name is <XML> then the Entity-Name MUST use an [AS].

Close
The close production marks the end of a XML element.

XML Encoding
In different contexts certain characters are reserved, for example / can not be used in an XML element name and > / < can not be used in a value. As such these values must be encoded as follows:

Encoding = Decimal | Hex4

Decimal = & Non-Zero *(0 | Non-Zero)

Hex4 = &u- 4(Hex)

Non-Zero = 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Hex = 0 | Non-Zero | A | B | C | D | E | F

The numbers MUST map to the UTF8 character encodings. Please note that the & character must always be encoded.

Markup Modifier
The markup modifier, (- , after the end of an XML element) instructs the principal how to treat a XML element with an unknown name. If the modifier is used and the XML element is not recognized then the XML element name MUST be stripped and the XML element s contents promoted one level in the parse tree. If the modifier is not used and the XML element name is unknown then the XML element and its contents MUST be ignored.

XML Syntax Shorthand
The following template is recommended for efficiently providing a description of an XML element.

Name: The name of the XML element

Purpose: A one line description of the XML element s purpose.

Schema: The schema, if any, that the XML element belongs to

Parent: XML elements that this XML element may be a child of.

Values: A description, usually a BNF, of the simple and compound values that the XML element supports.

Description: Further information.

Example: An example of the XML element s use.

Appendix 2 - Parameter Syntax for Content-Type Application/XML
HTTP 1.1 provides for a mechanism to include a parameter with a content type. In order to prevent namespace collisions the parameters for application/XML must use the following syntax:

Parameter = #(< >URI< > [= (token | Quoted-String)])

Schema Content-Type Parameter
Parameter = < > http://www.w3.org/standards/xml/content-type/schema < > = < > #URI < >

The http://www.w3.org/standards/xml/content-type/schema/ URL is used as a parameter to an application/xml content type. The URL indicates that its value lists some subset of the schemas defined in NameSpace parameters within the enclosed document. The URI can also be used in requests to indicate schemas that should appear in the result.

An example of the use of this parameter is to include it in an accept-type header on a request to indicate that the response should contain the specified namespace. Thus the client is able to inform the server of its support for a particular set of namespaces. The server is not required to return a result with the specified namespaces.

Appendix 3 URI Path Encoding

Problem Definition
A mechanism is needed to refer to specific DAV properties in a manner that can handle simple, composite, and multivalued DAV properties.

Solution Requirement
The reference mechanism must use the standard URL syntax so it can be used with both currently existing and future URLs. For example, the syntax could be appended to an HTTP URL to specify a HTTP property on that URL.

Path Component
URIPath = / [segment]

Segment = ((Abs-URI) | Rel-URI)[Index]*(; param)

Index = [([-]*digit)]

Abs-URI = < An absolute or relative URI which has been URI-Path encoded >

Rel-URI = < A relative URI for which URI-Encoding(Rel-URI) == Rel-URI >

URI-Path encoding consists of the following algorithm:

URL encode all ! characters

Map all / characters to ! characters

Please note that all relative URIs are relative to the URI

in the path segment preceding them. Hence the URI in the first path segment MUST be an absolute URI.

The purpose of the encoding is to allow URLs to be used as segments without having to use % encoding on all the / which produces a URL form which is extremely difficult for humans to deal with, and which changes the semantics of the URL.

Appendix 4 - XML URI

The XML scheme is to be registered with IANA as a reserved namespace that will be owned by the XML group through the W3C.

The new URI is defined as:

XML = XML : XML-Path

Appendix 5 - XML elements

Ref XML element
Name: XML:Ref

Purpose: A XML element that indicates that its contents are a URI.

Schema: XML

Parent: Any

Value = URI

Namespace

Namespace XML element

Name: XML:Namespace

Purpose: To inform the parser that a particular schema is in use and to provide a shorthand name for referring to XML elements related to that schema.

Schema: XML

Parent: Any

Value = (Ref [AS])

Description: This XML element contains two XML elements, Ref and AS. The purpose of the XML element is to inform the parser that a schema, identified by the value of the Ref XML element, is in use and, when appropriate, to provide a shorthand name to refer XML elements derived from that schema using the AS XML element. The AS mechanism is needed for efficiency reasons and because a URI can not be fully specified in an XML open tag. The Namespace XML element s scope is all siblings and their children.

AS XML element

Name: XML:AS

Purpose: To provide a short name for the URI of the schema provided in the Ref XML entity of a namespace XML entity.

Schema: XML

Parent: XML:Namespace

Value = 1*Alpha

Description: The AS XML entity is used to provide a shorthand reference for the URI in the Ref XML entity of a Namespace XML entity. The value contained in the AS XML entity is generated at the XML producer s discretion, the only requirement is that all AS values MUST be unique within the contents of the parent of the namespace element.

All XML entity open tags contain a name of the form As-Tag:Name. The As-Tag is the value defined in an AS XML entity inside of a Namespace. To resolve the As-Tag:Name into a properly formatted URI replace As-Tag: with the URI provided in the Ref that the AS was defined with. Also note that AS value also applies to any URIs defined in a Ref inside of Namespace.

For example,

```
     <XML>
          <XML:Namespace><Ref>http://blah;DAV/</><AS>B</></>
          <XML:Namespace><Ref>B:(B:)/</><AS>C</></>
          <C:Moo></>
     </>
```

So B:(B:) translates to http://blah;DAV/(http:!!blah;DAV!)/ and C:Moo translates to http://blah;DAV/(http:!!blah;DAV!)/Moo.

Required XML element

Name: XML:Required

Purpose: To indicate that the read MUST understand the associated Namespace in order to successfully process the XML document.

Schema: XML

Parent: XML:Namespace

Value: None

The XML URI and Namespace

In order to prevent a logical loop the XML namespace is said to be declared, with the AS value of XML as a consequence of the <XML> enclosing property.