Web Distributed Authoring and Versioning (WebDAV) Locking Protocol
draft-reschke-webdav-locking-05

Status of this Memo

By submitting this Internet-Draft, I certify that any applicable patent or other IPR claims of which I am aware have been disclosed, and any of which I become aware will be disclosed, in accordance with RFC 3668.

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 obsoleted 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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on January 12, 2005.

Copyright Notice

Copyright (C) The Internet Society (2004). All Rights Reserved.

Abstract

This document specifies a set of methods and headers ancillary to HTTP/1.1 (RFC2616) and Distributed Authoring and Versioning (WebDAV, RFC2518) for the management of resource locking (collision avoidance). It updates those sections from RFC2518 that specify WebDAV's locking features.

Editorial Note
Note that this document is not a product of the WebDAV
working group.

It is just an experiment to study the feasability of extracing the locking feature into a separate specification. --reschke]

Distribution of this document is unlimited. Please send comments to the WebDAV working group at <mailto:w3c-dist-auth@w3.org>, which may be joined by sending a message with subject "subscribe" to <mailto:w3c-dist-auth-request@w3.org?subject=subscribe>.

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

Table of Contents

1. Introduction ........................................... 5
  1.1 Terminology ........................................ 5
  1.2 Method Preconditions and Postconditions .......... 5
2. Overview of Locking ..................................... 6
  2.1 Exclusive Vs. Shared Locks ......................... 6
  2.1.1 Lock Compatibility Table ....................... 7
  2.2 Required Support ................................... 8
  2.3 Lock Tokens ......................................... 8
  2.4 Lock Capability Discovery ........................ 9
  2.5 Status of a lock ................................... 9
  2.5.1 Lock Access Type ................................ 9
  2.5.2 Lock Scope ...................................... 9
  2.5.3 Lock Root ........................................ 9
  2.5.4 Lock Depth ...................................... 9
  2.5.5 Lock Owner ..................................... 10
  2.5.6 Lock Timeout ................................... 10
  2.6 Active Lock Discovery .............................. 11
  2.7 Usage Considerations ............................... 11
3. Write Lock ............................................... 12
  3.1 Methods Restricted by Write Locks ................ 12
  3.2 Write Locks and Properties ......................... 12
  3.3 Write Locks and Collections ....................... 12
  3.4 Write Locks and the If Request Header ............ 13
  3.4.1 Example - Write Lock ........................... 14
  3.5 Write Locks and COPY/MOVE ........................ 14
4. Properties ................................................ 14
  4.1 Common XML elements used in property values ........ 15
  4.1.1 Lock Scopes .................................... 15
  4.1.2 Lock Types ..................................... 15
  4.2 DAV:lockdiscovery property ....................... 15
  4.2.1 Examples for the DAV:lockdiscovery ............. 16
  4.3 DAV:supportedlock property ....................... 17
  4.3.1 Examples for the DAV:supportedlock property .... 18
5. LOCK Method ............................................ 18
5.1 Creating Locks ............................................. 18
5.1.1 Marshalling ........................................... 18
5.1.2 Postconditions ........................................ 19
5.1.3 Example - Simple Lock Request ....................... 20
5.1.4 Example - Multi-Resource Lock Request ............... 22
5.2 Refreshing Locks ......................................... 23
5.2.1 Marshalling ........................................... 23
5.2.2 Preconditions ......................................... 23
5.2.3 Postconditions ........................................ 23
5.2.4 Example - Refreshing a Write Lock ................... 24
6. UNLOCK Method ............................................. 25
6.1 Marshalling ................................................ 25
6.2 Preconditions ............................................. 25
6.2.1 DAV:lock-token-matches precondition ................ 25
6.2.2 DAV:lock-removal-allowed precondition ............... 25
6.3 Postconditions ............................................ 26
6.3.1 DAV:lock-removed postcondition ...................... 26
6.4 Example - UNLOCK .......................................... 26
7. Additional status codes ................................. 26
7.1 423 Locked ................................................ 26
8. Additional marshalling and method semantics for other
methods .......................................................... 26
8.1 Additional marshalling .................................... 26
8.1.1 DAV:name-allowed precondition ....................... 27
8.1.2 DAV:parent-resource-must-be-non-null precondition .. 27
8.2 Additional method semantics .............................. 27
8.2.1 DAV:lock-token-submission-allowed precondition ... 27
8.2.2 DAV:need-lock-token precondition .................... 27
9. Headers ....................................................... 28
9.1 Lock-Token request/response header ...................... 28
9.2 Timeout request header .................................... 29
10. Capability discovery ....................................... 29
10.1 OPTIONS method .......................................... 29
11. Security considerations .................................... 30
11.1 Privacy Issues Connected to Locks ....................... 30
12. Internationalization Considerations ...................... 30
13. IANA Considerations ....................................... 30
14. Acknowledgements .......................................... 30
15. References .................................................. 31
15.1 Normative References .................................... 31
15.2 Informative References ................................... 31
16. Author’s Address ........................................... 32
A. Changes to RFC2518 ......................................... 32
A.1 Removed/Deprecated features ............................. 32
A.1.1 Implicit lock refresh ................................. 32
A.1.2 Lock-null resources .................................. 32
A.2 Additional features ....................................... 33
A.2.1 DAV:lockroot element in DAV:activelock .......... 33
A.2.2 Error Marshalling .................. 33
B. Text to be integrated from RFC2518 ............. 33
   B.1 HTTP Methods for Distributed Authoring ....... 33
   B.1.1 LOCK Method .......................... 33
   B.2 HTTP Headers for Distributed Authoring ....... 34
   B.2.1 If Header ............................ 34
C. GULP ........................................ 34
   C.1 Directly vs Indirectly ..................... 34
   C.2 Creating Locks ........................... 34
   C.3 Lock Inheritance ......................... 34
   C.4 Removing Locks ........................... 34
   C.5 Submitting Lock Tokens ..................... 34
   C.6 Locked State ............................. 35
   C.7 URL protection .......................... 35
   C.8 Exclusive vs Shared ...................... 35
D. ‘opaqueunlocktoken’ URI Scheme .................. 35
   D.1 Node Field Generation Without the IEEE 802 Address .... 36
E. Change Log (to be removed by RFC Editor before publication) .... 38
   E.1 Since draft-reschke-webdav-locking-00 ........ 38
   E.2 Since draft-reschke-webdav-locking-01 ........ 38
   E.3 Since draft-reschke-webdav-locking-02 ........ 38
   E.4 Since draft-reschke-webdav-locking-03 ........ 39
   E.5 Since draft-reschke-webdav-locking-04 ........ 39
F. Resolved issues (to be removed by RFC Editor before publication) .... 39
   F.1 040_LOCK_ISSUES_06 ........................ 39
   F.2 063_LOCKS_SHOULD_THEY_USE_AN_IF_HEADER_TO_VERIFY .... 40
   F.3 088_DAVOWNER_FIELD_IS_CLIENT_CONTROLED ........ 41
G. Open issues (to be removed by RFC Editor prior to publication) .... 41
   G.1 import-gulp ............................ 41
   G.2 edit ..................................... 41
   G.3 099_COPYMOVE_LOCKED_STATUS_CODE_CLARIFICATION ...... 41
   G.4 100_COPYMOVE_LOCKED_STATUS_DESCRIPTION ......... 42
   G.5 066_MUST_AN_IF_HEADER_CHECK_THE_ROOT_OF_URL ....... 42
Index ........................................... 43

Intellectual Property and Copyright Statements ........ 45
1. Introduction

This document describes the "locking" extension to the Web Distributed Authoring and Versioning (WebDAV) protocol that allows to keep more than one person from working on a document at the same time. This helps preventing the "lost update problem," in which modifications are lost as first one author then another writes changes without merging the other author’s changes.

1.1 Terminology

The terminology used here follows and extends that in the WebDAV Distributed Authoring Protocol specification [RFC2518].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

This document uses XML DTD fragments ([XML]) as a purely notational convention. WebDAV request and response bodies cannot be validated due to the specific extensibility rules defined in section 23 of [RFC2518] and due to the fact that all XML elements defined by this specification use the XML namespace name "DAV:". In particular:

- Element names use the "DAV:" namespace.
- Element ordering is irrelevant.
- Extension elements/attributes (elements/attributes not already defined as valid child elements) may be added anywhere, except when explicitly stated otherwise.

1.2 Method Preconditions and Postconditions

A "precondition" of a method describes the state on the server that must be true for that method to be performed. A "postcondition" of a method describes the state on the server that must be true after that method has completed. If a method precondition or postcondition for a request is not satisfied and unless a more specific HTTP status code applies, the response status of the request MUST be either 403 (Forbidden) if the request should not be repeated because it will always fail, or 409 (Conflict) if it is expected that the user might be able to resolve the conflict and resubmit the request.
In order to allow better client handling of error responses, a distinct XML element type is associated with each method precondition and postcondition of a request. When a particular precondition is
not satisfied or a particular postcondition cannot be achieved, the appropriate XML element MUST be returned as the child of a top-level DAV:,error element in the response body, unless otherwise negotiated by the request. In a 207 Multi-Status response, the DAV: error element would appear in the appropriate DAV: response description element.

2. Overview of Locking

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

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

2.1 Exclusive Vs. Shared Locks

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

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

With shared locks there are two trust sets that affect a resource. The first trust set is created by access permissions. Principals who are trusted, for example, may have permission to write to the resource. Among those who have access permission to write to the resource, the set of principals who have taken out a shared lock also must trust each other, creating a (typically) smaller trust set within the access permission write set.

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

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

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

2.1.1 Lock Compatibility Table

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

<table>
<thead>
<tr>
<th>Current lock state / Lock request</th>
<th>Shared Lock</th>
<th>Exclusive Lock</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>True</td>
<td>True</td>
</tr>
<tr>
<td>Shared Lock</td>
<td>True</td>
<td>False</td>
</tr>
<tr>
<td>Exclusive Lock</td>
<td>False</td>
<td>False*</td>
</tr>
</tbody>
</table>

Legend: True = lock may be granted. False = lock MUST NOT be
granted. *=It is illegal for a principal to request the same lock
twice.

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.

2.2 Required Support

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

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

2.3 Lock Tokens

A lock token is a type of state token, represented as a URI, which identifies a particular lock (see [RFC2518], section 9.4, for a definition of state tokens). A lock token is returned by every successful LOCK operation in the Lock-Token response header. The lock token also appears in the value of the DAV:lockdiscovery property, the value of which is returned in the body of the response to a successful LOCK operation (note that this property also includes the tokens of other current locks on the resource).

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

This specification provides a lock token URI scheme called "opaquelocktoken" that meets the uniqueness requirements. However servers are free to return any URI scheme so long as it meets the uniqueness requirements. Note that only URI schemes registered by the IETF can ensure uniqueness.

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

Since server lock support is optional, a client trying to lock a resource on a server can either try the lock and hope for the best, or perform some form of discovery to determine what lock capabilities the server supports. This is known as lock capability discovery. Lock capability discovery differs from discovery of supported access control types, since there may be access control types without corresponding lock types. A client can determine what lock types the server supports by retrieving the DAV:supportedlock property defined in Section 4.3.

2.5 Status of a lock

A lock is identified by a URI (the lock token URI) but in general, it does not have a HTTP URL, and thus can not be directly manipulated using HTTP methods. Instead, this specification defines the new methods LOCK (creating and refreshing locks, see Section 5) and UNLOCK (removing locks, see Section 6) that act indirectly on locks.

A lock has state that can be indirectly observed by using the DAV:lockdiscovery property defined in Section 4.2. At a minimum, the state of a lock consists of the items defined in the sections below. After lock creation, all parts of the state with the exception of the timeout value are immutable.

2.5.1 Lock Access Type

At present, this specification only defines one lock access type, the "write" lock defined in Section 3.

2.5.2 Lock Scope

A lock has either exclusive or shared scope (see Section 2.1).

2.5.3 Lock Root

A lock is created as effect of a LOCK (creation) method request. The lock root is the URL to which this request was adressed.

2.5.4 Lock Depth

A "depth 0" lock only affects the resource to which the LOCK request was adressed to (the lock root). This resource is said to be
"directly locked" by the lock.

On the other hand, a "depth infinity" lock on a collection additionally affects all members of that collection. These resources
are said to be "indirectly locked" by the lock. A "depth infinity" lock on a non-collection resource behaves exactly the same way as a "depth 0" lock.

2.5.5 Lock Owner

Clients can submit information about the lock owner when creating a lock. This information should be sufficient for either directly contacting a principal (such as a telephone number or email URI), or for discovering the principal (such as the URL of a homepage).

Owner information is kept with the lock so that it can be returned in the DAV:lockdiscovery property upon request. Note that this information is entirely client-controlled, thus a server MUST store the information faithfully just like if it appeared in a WebDAV dead property (see [RFC2518], section 4).

2.5.6 Lock Timeout

In general, a lock expires after a certain amount of time. This time can be specified in the LOCK creation request (however servers are not required to honor this request).

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

The timers used for timeout expiry can be reset by the client by submitting a LOCK refresh request.

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

A client MUST NOT assume that just because the time-out has expired the lock has been lost. Clients MUST assume that locks may
arbitrarily disappear at any time, regardless of the value given in
the Timeout header. The Timeout header only indicates the behavior of the server if "extraordinary" circumstances do not occur. For example, an administrator may remove a lock at any time or the system may crash in such a way that it loses the record of the lock’s existence.

2.6 Active Lock Discovery

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

2.7 Usage Considerations

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

- Two clients A and B are interested in editing the resource ‘index.html’. Client A is an HTTP client rather than a WebDAV client, and so does not know how to perform locking.
- Client A doesn’t lock the document, but does a GET and begins editing.
- Client B does LOCK, performs a GET and begins editing.
- Client B finishes editing, performs a PUT, then an UNLOCK.
- Client A performs a PUT, overwriting and losing all of B’s changes.

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

WebDAV servers that support locking can reduce the likelihood that clients will accidentally overwrite each other’s changes by requiring
clients to lock resources before modifying them. Such servers would effectively prevent HTTP 1.0 and HTTP 1.1 clients from modifying resources.
WebDAV clients can be good citizens by using a lock/retrieve/write/unlock sequence of operations (at least by default) whenever they interact with a WebDAV server that supports locking.

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

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

3. Write Lock

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

3.1 Methods Restricted by Write Locks

If a request would modify the content for a locked resource, a dead property of a locked resource, a live property that is defined to be lockable for a locked resource, or an internal member URI of a locked collection, the request MUST fail unless the lock-token for that lock is submitted in the request. An internal member URI of a collection is considered to be modified if it is added, removed, or identifies a different resource. [[anchor14: Copy of GULP, "Locked State".--reschke]]

3.2 Write Locks and Properties

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

3.3 Write Locks and Collections

A write lock on a collection, whether created by a "Depth: 0" or "Depth: infinity" lock request, prevents the addition or removal of member URIs of the collection by non-lock owners. As a consequence, when a principal issues a PUT or POST request to create a new resource under a URI which needs to be an internal member of a write locked collection to maintain HTTP namespace consistency, or issues a DELETE to remove an internal member URI of a write locked collection,
this request MUST fail if the principal does not have a write lock on the collection.
However, if a write lock request is issued to a collection containing member URIs identifying resources that are currently locked in a manner which conflicts with the write lock, the request MUST fail with a 423 (Locked) status code (Note that this can only occur for a request of a "Depth: infinity" write lock).

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

3.4 Write Locks and the If Request Header

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

In order to prevent these collisions a lock token MUST be submitted in the If header for all locked resources that a method may interact with or the method MUST fail. For example, if a resource is to be moved and both the source and destination are locked then two lock tokens must be submitted, one for the source and the other for the destination.

Servers SHOULD restrict usage of the lock token to exactly the authenticated principal who created the lock.
3.4.1 Example - Write Lock

>>Request

COPY /~fielding/index.html HTTP/1.1
Host: example.com
Destination: http://example.com/users/f/fielding/index.html
If: <http://example.com/users/f/fielding/index.html>
    (<opaquenotification:f81d4fae-7dec-11d0-a765-00a0c91e6bf6>)

>>Response

HTTP/1.1 204 No Content

In this example, even though both the source and destination are locked, only one lock token must be submitted, for the lock on the destination. This is because the source resource is not modified by a COPY, and hence unaffected by the write lock. In this example, user agent authentication has previously occurred via a mechanism outside the scope of the HTTP protocol, in the underlying transport layer.

3.5 Write Locks and COPY/MOVE

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

A successful MOVE request on a write locked resource MUST NOT move the write lock with the resource. However, the resource is subject to being added to an existing lock at the destination, as specified in Section 3.3. For example, if the MOVE makes the resource a child of a collection that is locked with "Depth: infinity", then the resource will be added to that collection’s lock. Additionally, if a resource locked with "Depth: infinity" is moved to a destination that is within the scope of the same lock (e.g., within the namespace tree covered by the lock), the moved resource will again be added to the lock. In both these examples, as specified in Section 3.4, an If header must be submitted containing a lock token for both the source and destination.

4. Properties

Any DAV compliant resource that supports the LOCK method MUST support the DAV:activelock and DAV:lockdiscovery properties defined below.
4.1 Common XML elements used in property values

4.1.1 Lock Scopes

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

4.1.2 Lock Types

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

4.2 DAV:lockdiscovery property

The DAV:lockdiscovery property returns a listing of who has a lock, what type of lock he has, the timeout type, the time remaining on the timeout, the associated lock token and the root of the lock. The server is free to withhold any or all of this information if the requesting principal does not have sufficient access rights to see the requested data.

<!ELEMENT lockdiscovery (activelock)* >

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

depth: the value of the Depth header (see Section 2.5.4; takes the values "0" or "infinity").

<!ELEMENT depth (#PCDATA) >

owner: provides information about the principal taking out a lock (see Section 2.5.5).

<!ELEMENT owner ANY>

timeout: the time remaining until timeout of a lock (see Section 2.5.6).
locktoken: the lock token associated with a lock; the href element contains the lock token.
lockroot: the URL that was specified as Request-URI in the LOCK creation request; the href element contains the URL (see Section 2.5.3).

hhref: defined in [RFC2518], section 12.3.

4.2.1 Examples for the DAV:lockdiscovery
DAV:lockdiscovery property for a resource that has two shared write locks on it, with infinite timeouts:

```xml
<D:lockdiscovery xmlns:D="DAV:">
  <D:activelock>
    <D:locktype><D:write/></D:locktype>
    <D:lockscope><D:shared/></D:lockscope>
    <D:depth>0</D:depth>
    <D:owner>Jane Smith</D:owner>
    <D:timeout>Infinite</D:timeout>
    <D:locktoken>
      <D:href>opaque:\locktoken:f81de2ad-7f3d-a1b2-4f3c-00a0c91a9d76</D:href>
    </D:locktoken>
    <D:lockroot>
      <D:href>http://example.com/container/</D:href>
    </D:lockroot>
  </D:activelock>
</D:lockdiscovery>

<D:lockdiscovery>
  <D:activelock>
    <D:locktype><D:write/></D:locktype>
    <D:lockscope><D:shared/></D:lockscope>
    <D:depth>0</D:depth>
    <D:owner>John Doe</D:owner>
    <D:timeout>Infinite</D:timeout>
    <D:locktoken>
      <D:href>opaque:\locktoken:f81de2ad-7f3d-a1b2-4f3c-00a0c91a9d77</D:href>
    </D:locktoken>
    <D:lockroot>
      <D:href>http://example.com/container/</D:href>
    </D:lockroot>
  </D:activelock>
</D:lockdiscovery>
```

DAV:lockdiscovery property for a resource with no locks on it:

```xml
<D:lockdiscovery xmlns:D="DAV:">
</D:lockdiscovery>
```

4.3 DAV:supportedlock property

The DAV:supportedlock property of a resource returns a listing of the combinations of scope and access types which may be specified in a lock request on the resource. Note that the actual contents are
themselves controlled by access controls so a server is not required
to provide information the client is not authorized to see.

<!ELEMENT supportedlock (lockentry)* >
<!ELEMENT lockentry (lockscope, locktype)> 

4.3.1 Examples for the DAV:supportedlock property

DAV:supportedlock property for a resource that supports both exclusive and shares write locks:

<D:supportedlock xmlns:D="DAV:"
            <D:lockentry>
                <D:lockscope><D:exclusive/></D:lockscope>
                <D:locktype><D:write/></D:locktype>
            </D:lockentry>
            <D:lockentry>
                <D:lockscope><D:shared/></D:lockscope>
                <D:locktype><D:write/></D:locktype>
            </D:lockentry>
        </D:supportedlock>

DAV:supportedlock property for a resource that doesn’t support any locks at all:

<D:supportedlock xmlns:D="DAV:"/>

5. LOCK Method

The following sections describe the LOCK method, which is used to take out a lock of any access type or to refresh an existing lock.

5.1 Creating Locks

A LOCK method invocation with non-empty request body creates the lock specified by the lockinfo XML element on the resource identified by the Request-URI. If the Request-URI identifies a null resource, the invocation MUST create a new resource with empty content.

5.1.1 Marshalling

The request MAY include a "Timeout" header to be used as the timeout
value for the lock to be created (see Section 9.2). However, the server is not required to honor or even consider this request.

The request MAY include a "Depth" header specifying either "0" or
"infinity" (see [RFC2518], section 9.2). If no "Depth" header is submitted then the request MUST act as if a "Depth:infinity" had been specified.

The request body MUST be a DAV:lockinfo element:

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

DAV:lockscope, DAV:locktype and DAV:owner are defined in Section 4.

The response body for a successful request MUST be a DAV:prop XML element, containing the new value for the DAV:lockdiscovery property defined in Section 4.2. The lock token URI for the new lock MUST be returned in the "Lock-Token" response header (see Section 9.1).

[[anchor25: Add preconditions for the validy of various parts of the request body? --reschke]]

5.1.2 Postconditions

5.1.2.1 DAV:create-lock postcondition

The request MUST have created a new lock on the resource identified by the Request-URI. The request MUST have allocated a distinct new lock token URI for the new lock, and that URI MUST NOT ever identify anything other than that lock. [[anchor28: Say what parts of the request lock criteria must be followed. --reschke]]

5.1.2.2 DAV:create-resource postcondition

If the request-URI identified a null resource, the method MUST have created a new resource with empty content.
5.1.3 Example - Simple Lock Request

>>Request

LOCK /workspace/webdav/proposal.doc HTTP/1.1
Host: example.com
Timeout: Infinite, Second-4100000000
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx
Authorization: Digest username="ejw",
    realm="ejw@example.com", nonce="...",
    uri="/workspace/webdav/proposal.doc",
    response="...", opaque="...

<?xml version="1.0" encoding="utf-8" ?>
<D:lockinfo xmlns:D='DAV:'>
    <D:lockscope><D:exclusive/></D:lockscope>
    <D:locktype><D:write/></D:locktype>
    <D:owner>
        <D:href>http://example.org/~ejw/contact.html</D:href>
    </D:owner>
</D:lockinfo>
HTTP/1.1 200 OK
Lock-Token: <opaquelocktoken:e71d4fae-5dec-22d6-fecha5-00a0c91e6be4>
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:prop xmlns:D="DAV:"
   <D:lockdiscovery>
     <D:activelock>
       <D:locktype><D:write/></D:locktype>
       <D:lockscope><D:exclusive/></D:lockscope>
       <D:depth>Infinity</D:depth>
       <D:owner>
         <D:href>http://example.org/~ejw/contact.html</D:href>
       </D:owner>
       <D:timeout>Second-604800</D:timeout>
       <D:locktoken>
         <D:href>opaquelocktoken:e71d4fae-5dec-22d6-fecha5-00a0c91e6be4</D:href>
       </D:locktoken>
       <D:lockroot>
       </D:lockroot>
     </D:activelock>
   </D:lockdiscovery>
</D:prop>

This example shows the successful creation of an exclusive write lock on resource http://example.com/workspace/webdav/proposal.doc. The resource http://example.org/~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).
5.1.4 Example - Multi-Resource Lock Request

>>>Request

LOCK /webdav/ HTTP/1.1
Host: example.com
Timeout: Infinite, Second-4100000000
Depth: infinity
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx
Authorization: Digest username="ejw",
    realm="ejw@example.com", nonce="...",
    uri="/workspace/webdav/proposal.doc",
    response="...", opaque="..."

<?xml version="1.0" encoding="utf-8" ?>
<D:lockinfo xmlns:D="DAV:">
    <D:locktype><D:write/></D:locktype>
    <D:lockscope><D:exclusive/></D:lockscope>
    <D:owner>
        <D:href>http://example.org/~ejw/contact.html</D:href>
    </D:owner>
</D:lockinfo>

>>>Response

HTTP/1.1 207 Multi-Status
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:">
    <D:response>
        <D:href>/webdav/secret</D:href>
        <D:status>HTTP/1.1 403 Forbidden</D:status>
    </D:response>
    <D:response>
        <D:href>/webdav/</D:href>
        <D:status>HTTP/1.1 424 Failed Dependency</D:status>
    </D:response>
</D:multistatus>

This example shows a request for an exclusive write lock on a collection and all its children. In this request, the client has specified that it desires an infinite length lock, if available, otherwise a timeout of 4.1 billion seconds, if available. The request entity body contains the contact information for the principal taking out the lock, in this case a web page URL.
The error is a 403 (Forbidden) response on the resource http://example.com/webdav/secret. Because this resource could not be locked, none of the resources were locked.

5.2 Refreshing Locks

A LOCK request with no request body is a "LOCK refresh" request. It’s purpose is to restart all timers associated with a lock.

If an error is received in response to a refresh LOCK request the client SHOULD assume that the lock was not refreshed. [[anchor32: This fact is so obvious that it should be removed. --reschke]]

5.2.1 Marshalling

The request MUST include an "If" header that contains the lock tokens of the locks to be refreshed (note there may be multiple in the case of shared locks).

The request MAY include a "Timeout" header to be used as the new timeout value for the lock(s) to be refreshed (see Section 9.2).

The response to a successful lock refresh request MUST contain the value of the current DAV:lockdiscovery property in a prop XML element.

5.2.2 Preconditions

5.2.2.1 DAV:lock-submission-allowed precondition

See Section 8.2.1.

5.2.3 Postconditions

5.2.3.1 DAV:locks-refreshed postcondition

Timers associated with the those locks submitted in the "If" request header whose lock root is the resource identified by the Request-URI MUST be reset to their original value (or alternatively to the new value given in the "Timeout" request header).
5.2.4 Example - Refreshing a Write Lock

>>Request

LOCK /workspace/webdav/proposal.doc HTTP/1.1
Host: example.com
Timeout: Infinite, Second-4100000000
If: (opaque:locktoken:e71d4fae-5dec-22d6-f3a5-00a0c91e6be4)
Authorization: Digest username="ejw",
realm="ejw@example.com", nonce="...",
uri="/workspace/webdav/proposal.doc",
response="...", opaque="..."

>>Response

HTTP/1.1 200 OK
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:prop xmlns:D="DAV:"
    <D:lockdiscovery>
      <D:activelock>
        <D:locktype><D:write/></D:locktype>
        <D:lockscope><D:exclusive/></D:lockscope>
        <D:depth>Infinity</D:depth>
        <D:owner>
          <D:href>http://example.org/~ejw/contact.html</D:href>
        </D:owner>
        <D:timeout>Second-604800</D:timeout>
        <D:locktoken>
          <D:href>opaque:locktoken:e71d4fae-5dec-22d6-f3a5-00a0c91e6be4</D:href>
        </D:locktoken>
        <D:lockroot>
        </D:lockroot>
      </D:activelock>
    </D:lockdiscovery>
</D:prop>

This request would refresh the lock, resetting any time outs. Notice that the client asked for an infinite time out but the server choose to ignore the request.
6. UNLOCK Method

The UNLOCK method removes the lock identified by the lock token in the Lock-Token request header from the resource identified by the Request-URI, and all other resources included in the lock. Note that the UNLOCK request may be submitted to any resource locked by that lock (even those that are locked indirectly).

If all resources which have been locked under the submitted lock token can not be unlocked then the UNLOCK request MUST fail.

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

A server MAY allow principals other than a lock owner to unlock a resource. In this case, this capability SHOULD be under access control (see [RFC3744], section 3.5). Note that there is a tradeoff in allowing non-owners of a lock to unlock a resource. It can be beneficial to allow non-lock owners to perform UNLOCK requests because it allows the administrator of the server to configure the server to grant longer lock timeouts because the administrator knows that there is a process in place to allow users to deal with forgotten locks left by other users. On the other hand, a disadvantage of unlocking someone else’s lock is that can create a situation where two users are working on modifications to the same resource at the same time which can result in a client having to perform an merge that wasn’t previously planned.

6.1 Marshalling

The request MUST include a "Lock-Token" header (see Section 9.1) that identifies the lock to be removed.

6.2 Preconditions

6.2.1 DAV:lock-token-matches precondition

The lock identified by the "Lock-Token" request header exists, and the resource identified by the Request-URI indeed is directly locked by the specified lock.

6.2.2 DAV:lock-removal-allowed precondition
As discussed above, the principal authenticated for the UNLOCK request MUST be allowed to remove the identified lock (note that servers that support the "WebDAV Access Control Protocol" should use the
DAV:need-privileges precondition defined in section 7.1.1 of [RFC3744]).

6.3 Postconditions

6.3.1 DAV:lock-removed postcondition

The lock MUST have been removed from all resources included in the lock.

6.4 Example - UNLOCK

>>Request

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

>>Response

HTTP/1.1 204 No Content

In this example, the lock identified by the lock token
"opaquelocktoken:a515cfa4-5da4-22e1-f5b5-00a0451e6bf7" is
successfully removed from the resource
http://example.com/workspace/webdav/info.doc. If this lock included
more than just one resource, the lock is removed from all resources
included in the lock. Note that clients MUST interpret any of the
success status codes defined in [RFC2616], section 10.2 as success
codes. 204 (No Content) was used here merely for consistency with
the example in [RFC2518], section 8.11.1).

7. Additional status codes

7.1 423 Locked

The 423 (Locked) status code means the source or destination resource
of a method is locked.
8. Additional marshalling and method semantics for other methods

8.1 Additional marshalling

This section defines additional condition names (see Section 1.2)
that apply to all methods.

8.1.1 DAV:name-allowed precondition

If a request such as COPY, LOCK, MOVE, PUT or MKCOL is going to create a new internal member URI inside a collection resource, the last path segment of that URI must specify a name that is available as a resource name (for instance, servers may disallow path segments that -- after being URI-unescaped -- aren’t valid UTF-8 octet sequences).  

[anchor50: Copied from draft-ietf-webdav-redirectref-protocol. --reschke]

8.1.2 DAV:parent-resource-must-be-non-null precondition

If a request such as COPY, LOCK, MOVE, PUT or MKCOL is going to create a new internal member URI inside a collection resource, that collection resource must be non-null.  

[anchor52: Copied from draft-ietf-webdav-redirectref-protocol. --reschke]

8.2 Additional method semantics

This section defines names (see Section 1.2) for new conditions introduced by locking semantics. Otherwise noted otherwise, they apply to all methods.

8.2.1 DAV:lock-token-submission-allowed precondition

If the server restricts usage of the lock token inside an "If" request header to specific principals, the authenticated principal for this request MUST be one of them.

8.2.2 DAV:need-lock-token precondition

If a request would modify the content for a locked resource, a dead property of a locked resource, a live property that is defined to be lockable for a locked resource, or an internal member URI of a locked collection, the request MUST fail unless the lock-token for that lock is submitted in the request. An internal member URI of a collection is considered to be modified if it is added, removed, or identifies a different resource.  

[anchor55: Copied from GULP. --reschke]

<!ELEMENT need-lock-token (href)* >

Servers SHOULD insert DAV:href elements for the URLs of each root of
a lock for which a lock token was needed, unless that URL identifies the same resource to that the request was sent.
8.2.2.1 Example

In the example below, a client unaware of a "Depth: infinity" lock on the parent collection "/workspace/webdav/" attempts to modify the collection member "/workspace/webdav/proposal.doc".

>>Request

    PUT /workspace/webdav/proposal.doc HTTP/1.1
    Host: example.com

>>Response

    HTTP/1.1 423 Locked
    Content-Type: text/xml; charset="utf-8"
    Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:error xmlns:D="DAV:">
  <D:need-lock-token>
    <D:href>/workspace/webdav/</D:href>
  </D:need-lock-token>
</D:error>

9. Headers

9.1 Lock-Token request/response header

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

Note that the "Coded-URL" production is defined in [RFC2518], section 9.4.

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

The Lock-Token response header is used with the LOCK method to indicate the lock token created as a result of a successful LOCK request to create a new lock.
Note that the "Lock-Token" request header does not contribute to the precondition checks defined for the HTTP status 412 (see [RFC2616], section 10.4.13).
9.2 Timeout request header

Timeout = "Timeout" ":" 1#TimeType
TimeType = ("Second-" DAVTimeOutVal | "Infinite" | Other)
DAVTimeOutVal = 1*<digit>
Other = "Extend" field-value ; See section 4.2 of [RFC2616]

Clients MUST NOT submit a Timeout request header with any method other than a LOCK method.

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

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

The "Second" TimeType specifies the number of seconds that will elapse between granting of the lock at the server, and the automatic removal of the lock. The timeout value for TimeType "Second" MUST NOT be greater than 2^32-1.

10. Capability discovery

10.1 OPTIONS method

If the server supports locking, it MUST return both the compliance class names "2" and "locking" as fields in the "DAV" response header (see [RFC2518], section 9.1) from an OPTIONS request on any resource implemented by that server. A value of "2" or "locking" in the "DAV" response header MUST indicate that the server meets all class "1" requirements defined in [RFC2518] and supports all MUST level requirements and REQUIRED features specified in this document, including:

- LOCK and UNLOCK methods,
- DAV:lockdiscovery and DAV:supportedlock properties,
- "Time-Out" request header, "Lock-Token" request and response header.
Note that for servers implementing this specification, the compliance classes "2" and "locking" are synonymous. However, new clients can
take advantage of the new "locking" compliance class to detect server support for changes introduced by this specification (see Appendix A).

11. Security considerations

All security considerations mentioned in [RFC2518] also apply to this document. Additionally, lock tokens introduce new privacy issues discussed below.

11.1 Privacy Issues Connected to Locks

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

12. Internationalization Considerations

All internationalization considerations mentioned in [RFC2518] also apply to this document.

13. IANA Considerations

This specification updates the definition of the "opaquelocktoken" URI scheme described in Appendix D, registered my means of [RFC2518], section 6.4. There are no additional IANA considerations.

14. Acknowledgements

This document is the collaborative product of

- the authors,
- the maintainers of RFC2518bis - Jason Crawford and Lisa Dusseault - and
the original authors of RFC2518 - Steve Carter, Asad Faizi, Yaron Goland, Del Jensen and Jim Whitehead.
This document has also benefited from thoughtful discussion by Mark Anderson, Dan Brotsky, Geoff Clemm, Jim Davis, Stefan Eissing, Rickard Falk, Larry Masinter, Joe Orton, Juergen Pill, Elias Sinderson, Greg Stein, Kevin Wiggen, and other members of the WebDAV working group.

15. References

15.1 Normative References


15.2 Informative References


URIs

Appendix A. Changes to RFC2518

See Section 10 for a description about how clients can discover support for this version of the WebDAV Locking protocol.

A.1 Removed/Deprecated features

A.1.1 Implicit lock refresh

In section 9.8, [RFC2518] specifies that locks should be refreshed implicitly every time "...any time an owner of the lock sends a method to any member of the lock, including unsupported methods, or methods which are unsuccessful." This feature has been removed (locks need to be refreshed explicitly using the LOCK method).

Compatibility consideration: clients historically have never relied on this feature as it was never implemented in widely deployed WebDAV servers.

A.1.2 Lock-null resources

In section 7.4, [RFC2518] specifies a special resource type called "lock-null resource" that’s being created when a LOCK method request is applied to a null resource. In practice, no real interoperability was achieved because many servers failed to implement this feature properly and few clients (if any) ever relied on that particular functionality.

Removing this feature also means that there is no atomic way to create a collection in locked state, but in practice, this doesn’t seem to be a problem.
Compatibility consideration: there do not seem to be any widely deployed clients that actually relied on "lock-null resources".
A.2 Additional features

A.2.1 DAV:lockroot element in DAV:activelock

Clients can take advantage of the new DAV:lockroot element to discover the URL to which the LOCK request (that created the lock) was applied.

Compatibility consideration: clients will have to fail gracefully when communicating with older servers that do not support the new element.

A.2.2 Error Marshalling

Clients can take advantage of additional, detailed error information using the DAV:error element defined in Section 1.2.

Compatibility consideration: old clients should not even notice the additional informations. New clients SHOULD handle absence of additional error information gracefully.

Appendix B. Text to be integrated from RFC2518

B.1 HTTP Methods for Distributed Authoring

B.1.1 LOCK Method

B.1.1.1 Locking Replicated Resources

A resource may be made available through more than one URI. However locks apply to resources, not URIs. Therefore a LOCK request on a resource MUST NOT succeed if can not be honored by all the URIs through which the resource is addressable.

B.1.1.2 Depth and Locking

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

If the Depth header is set to infinity then the resource specified in the Request-URI along with all its internal members, all the way down the hierarchy, are to be locked. A successful result MUST return a
single lock token which represents all the resources that have been locked. If an UNLOCK is successfully executed on this token, all associated resources are unlocked. If the lock cannot be granted to all resources, a 207 (Multistatus) status code MUST be returned with a response entity body containing a multistatus XML element describing which resource(s) prevented the lock from being granted.
Hence, partial success is not an option. Either the entire hierarchy is locked or no resources are locked.

B.1.1.3 Interaction with other Methods

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

B.2 HTTP Headers for Distributed Authoring

B.2.1 If Header

[[anchor72: Add "If" header considerations: --reschke]]

Appendix C. GULP

*Copied from *[3].

C.1 Directly vs Indirectly

A lock either directly or indirectly locks a resource.

C.2 Creating Locks

A LOCK request with a non-empty body creates a new lock, and the resource identified by the Request-URI is directly locked by that lock. The "lock-root" of the new lock is the Request-URI. If at the time of the request, the Request-URI is not mapped to a resource, a new resource with empty content MUST be created by the request.

C.3 Lock Inheritance

If a collection is directly locked by a depth:infinity lock, all members of that collection (other than the collection itself) are indirectly locked by that lock. In particular, if an internal member resource is added to a collection that is locked by a depth:infinity lock, and if the resource is not locked by that lock, then the resource becomes indirectly locked by that lock. Conversely, if a resource is indirectly locked with a depth:infinity lock, and if the result of deleting an internal member URI is that the resource is no longer a member of the collection that is directly locked by that lock, then the resource is no longer locked by that lock.
C.4 Removing Locks

An UNLOCK request deletes the lock with the specified lock token. The request-URL of the request MUST identify a resource that is
either directly or indirectly locked by that lock. After a lock is deleted, no resource is locked by that lock.

C.5 Submitting Lock Tokens

A lock token is "submitted" in a request when it appears in an "If" request header.

C.6 Locked State

If a request would modify the content for a locked resource, a dead property of a locked resource, a live property that is defined to be lockable for a locked resource, or an internal member URI of a locked collection, the request MUST fail unless the lock-token for that lock is submitted in the request. An internal member URI of a collection is considered to be modified if it is added, removed, or identifies a different resource.

C.7 URL protection

If a request causes a directly locked resource to no longer be mapped to the lock-root of that lock, then the request MUST fail unless the lock-token for that lock is submitted in the request. If the request succeeds, then that lock MUST have been deleted by that request.

C.8 Exclusive vs Shared

If a request would cause a resource to be locked by two different exclusive locks, the request MUST fail.

Appendix D. 'opaquelocktoken' URI Scheme

The opaqulocktoken URI scheme is designed to be unique across all resources for all time. Due to this uniqueness quality, a client may submit an opaque lock token in an If header on a resource other than the one that returned it.

All resources MUST recognize the opaqulocktoken scheme and, at minimum, recognize that the lock token does not refer to an outstanding lock on the resource.

In order to guarantee uniqueness across all resources for all time the opaqulocktoken requires the use of the Universal Unique Identifier (UUID) mechanism, as described in [ISO-11578].
Opaquelocktoken generators, however, have a choice of how they create these tokens. They can either generate a new UUID for every lock token they create or they can create a single UUID and then add
extension characters. If the second method is selected then the
program generating the extensions MUST guarantee that the same
extension will never be used twice with the associated UUID.

OpaqueLockToken-URI = "opaquelocktoken:" UUID [Extension] ; The UUID
production is the string representation of a UUID, as defined in
[ISO-11578]. Note that white space (LWS) is not allowed between
elements of this production.

Extension = path ; path is defined in [RFC2396], section 3.3.

D.1 Node Field Generation Without the IEEE 802 Address

UUIDs, as defined in [ISO-11578], contain a "node" field that
contains one of the IEEE 802 addresses for the server machine. As
noted in Section 11.1, there are several security risks associated
with exposing a machine's IEEE 802 address. This section provides an
alternate mechanism for generating the "node" field of a UUID which
does not employ an IEEE 802 address. WebDAV servers MAY use this
algorithm for creating the node field when generating UUIDs. The
text in this section is originally from an Internet-Draft by Paul
Leach and Rich Salz, who are noted here to properly attribute their
work.

The ideal solution is to obtain a 47 bit cryptographic quality random
number, and use it as the low 47 bits of the node ID, with the most
significant bit of the first octet of the node ID set to 1. This bit
is the unicast/multicast bit, which will never be set in IEEE 802
addresses obtained from network cards; hence, there can never be a
conflict between UUIDs generated by machines with and without network
cards.

If a system does not have a primitive to generate cryptographic
quality random numbers, then in most systems there are usually a
fairly large number of sources of randomness available from which one
can be generated. Such sources are system specific, but often
include:

- the percent of memory in use
- the size of main memory in bytes
- the amount of free main memory in bytes
- the size of the paging or swap file in bytes
- free bytes of paging or swap file
- the total size of user virtual address space in bytes
- the total available user address space bytes
- the size of boot disk drive in bytes
- the free disk space on boot drive in bytes
- the current time
- the amount of time since the system booted
- the individual sizes of files in various system directories
- the creation, last read, and modification times of files in various system directories
- the utilization factors of various system resources (heap, etc.)
- current mouse cursor position
- current caret position
- current number of running processes, threads
- handles or IDs of the desktop window and the active window
- the value of stack pointer of the caller
- the process and thread ID of caller
- various processor architecture specific performance counters (instructions executed, cache misses, TLB misses)

(Note that it is precisely the above kinds of sources of randomness that are used to seed cryptographic quality random number generators on systems without special hardware for their construction.)

In addition, items such as the computer’s name and the name of the
operating system, while not strictly speaking random, will help differentiate the results from those obtained by other systems.

The exact algorithm to generate a node ID using these data is system specific, because both the data available and the functions to obtain them are often very system specific. However, assuming that one can concatenate all the values from the randomness sources into a buffer, and that a cryptographic hash function such as MD5 is available, then any 6 bytes of the MD5 hash of the buffer, with the multicast bit
(the high bit of the first byte) set will be an appropriately random
node ID.

Other hash functions, such as SHA-1, can also be used. The only
requirement is that the result be suitably random in the sense that
the outputs from a set uniformly distributed inputs are themselves
uniformly distributed, and that a single bit change in the input can
be expected to cause half of the output bits to change.

Appendix E. Change Log (to be removed by RFC Editor before publication)

E.1 Since draft-reschke-webdav-locking-00

Add and resolve issue "rfc2606-compliance". Resolve issues
"extract-locking", "updated-rfc2068", "022_COPY_OVERWRITE_LOCK_NULL",
"025_LOCK_REFRESH_BY_METHODS", "037_DEEP_LOCK_ERROR_STATUS",
"039_MISSING_LOCK_TOKEN", "040_LOCK_ISSUES_01", "040_LOCK_ISSUES_02",
"040_LOCK_ISSUES_05", "043_NULL_LOCK_SLASH_URL",
"065_UNLOCK_WHAT_URL", "077_LOCK_NULL_STATUS_CREATION",
"080_DEFER_LOCK_NULL_RESOURCES_IN_SPEC",
"089_FINDING_THE_ROOT_OF_A_DEPTH_LOCK",
"101_LOCKDISCOVERY_FORMAT_FOR_MULTIPLE_SHARED_LOCKS",
"109_HOW_TO_FIND_THE_ROOT_OF_A_LOCK" and
"111_MULTIPLE_TOKENS_PER_LOCK". Add issue "import-gulp". Start work
on moving text from RFC2518 excerpts into new sections. Define new
compliance class "locking" (similar to "bis" in RFC2518bis, but only
relevant to locking). Reformatted "GULP" into separate subsections
for easier reference.

E.2 Since draft-reschke-webdav-locking-01

Update "008_URI_URL", "040_LOCK_ISSUES_06",
"063_LOCKS_SHOULD THEY USE AN_IF_HEADER_TO_VERIFY",
"067_UNLOCK_NEEDS_IF_HEADER", "068_UNLOCK_WITHOUT_GOOD_TOKEN".
Re-opened "065_UNLOCK_WHAT_URL". Close
"070_LOCK_RENEWAL_SHOULD_NOT_USE_IF_HEADER". Rewrite UNLOCK and LOCK
refresh method descriptions. Fix page title (TXT version). Close
"052_LOCK_BODY_SHOULD_BE_MUST", "054_IF_AND_AUTH",
"060_LOCK_REFRESH_BODY" and "079_UNLOCK_BY_NON_LOCK_OWNER". Add and
resolve "8.10.1_lockdiscovery_on_failure". Started attempt to
clarify status code.

E.3 Since draft-reschke-webdav-locking-02

Resolve issues "040_LOCK_ISSUES_03", "040_LOCK_ISSUES_04",
"040_LOCK_ISSUES_08" "053_LOCK_INHERITANCE", "057_LOCK_SEMANTICS",
"067_UNLOCK_NEEDS_IF_HEADER" and "068_UNLOCK_WITHOUT_GOOD_TOKEN".
Resolve issue "065_UNLOCK_WHAT_URL"; update to new GULP version
(5.7). Add and resolve new issue "7.5_DELETE_vs_URIIs”. Start work on "additional marshalling" and "introduction”. Update issues "044_REPORT_OTHER_RESOURCE_LOCKED" and "066_MUST_AN_IF_HEADER_CHECK_THE_ROOT_OF_URL”.

E.4 Since draft-reschke-webdav-locking-03

Close issues "import-rfc3253-stuff", "008_URI_URL", "015_MOVE_SECTION_6.4.1_TO_APPX", "044_REPORT_OTHER_RESOURCE_LOCKED", "056DEPTH_LOCK_AND_IF" and "072_LOCK_URL_WITH_NO_PARENT_COLLECTION". Reformat condition name descriptions. Add mention of condition failure signalling to "Changes" appendix. Start edit of header descriptions (Depth, Timeout) and LOCK creation description. Open and close issue "3.2_lockdiscovery_depth". Start work on intro.

E.5 Since draft-reschke-webdav-locking-04

Add description of the lock as a resource and it’s state (merging in Timeout semantics from old headers section). Close issues "040_LOCK_ISSUES_06”, "063_LOCKS_SHOULD THEY_USE_AN_IF_HEADER_TO_VERIFY” and "088_DAVOWNER_FIELD_IS_CLIENT_CONTROLLED”. Move edited version of "Write Lock" chapter.

Appendix F. Resolved issues (to be removed by RFC Editor before publication)

Issues that were either rejected or resolved in this version of this document.

F.1 040_LOCK_ISSUES_06

Type: change

<http://lists.w3.org/Archives/Public/w3c-dist-auth/1999AprJun/0246.html>

ccjason@us.ibm.com (1999-06-07): Upon cursory reading of the rfc 2518 sec 8.10.4 through 8.11 I was confused by the plethora of error codes. Nothing seems to unify them. 8.10.4 speaks of a return code of 409 Conflict if a lock can’t be granted. - Firstly, I can’t tell if it is saying that the 409 is within the multistatus body... or in the response header. - Secondly, later text seems to use a different status codes and never mentions this one again. 8.10.7 lists status codes - 200 OK, 412 Precondition Failed, and 423 Locked are listed, but 409 Conflict (mentioned above) is not. - In the case of 412
Precondition Failed, the description the follows doesn’t seem to describe a "precondition failed". And it sounds like it’s talking
about an access request that includes a "locktoken", not a LOCK request that generates one. - The 423 Locked condition also sort of sounds like it's talking about an access request rather than a LOCK request. 8.10.10 lists LOCK status codes - 207 Multistatus which was not mentioned above - 403 Forbidden which was not mentioned above. - 424 Failed dependency which was not mentioned above. 8.11 UNLOCK - we don’t mention what the failure response should look like. - comment: 200 OK seems like a better response than 204 No Content. The brief explanation isn’t persuasive and seems to say that the response code should serve the purpose of the Content-Length. header. - we should probably explicitly say if an UNLOCK can only be done on the original resource... and will fail even if the resource specified is locked by virtue of being a child of the original resource. Or is this too obvious? I know it’s something easy to goof up in an implementation.

julian.reschke@greenbytes.de (2004-06-04): (1) 8.10.4 is wrong. The return code is 207. See issue 037_DEEP_LOCK_ERROR_STATUS, resolved in draft 01. (2) General agreement that descriptions of error marshalling needs to be redone. This applies both to LOCK and UNLOCK. (3) Agreement that the argument given for 204 is lame; clients should handle all 2xx status codes with the notable exception of 207 as "success". We may want to explain that in RFC2518bis’ section about 207.

Resolution (2004-07-10): Resolved after cleaning up error marshalling descriptions and rewriting the statement about UNLOCK and status 204.

F.2 063_LOCKS_SHOULD THEY_USE_AN_IF_HEADER_TO_VERIFY

Type: change

jrd3@alum.mit.edu (): Is the complexity of the IF header appropriate for the simple task of verifying that a client knowingly owns a lock? The IF header seems to serve a different purpose. One of those purposes is for the server to verify that you have the lock token (and that you know the root of it?). Another is for the client to check some preconditions before doing an action. Another seems to be to specify what lock to refresh in a lock refresh request. This seems to create ambiguity in our definition of the semantics of the IF: header.

ccjason@us.ibm.com (): It is felt by the group that it’s important that the client not just own and hold the lock token, but that it also know where the lock is rooted before it does tasks related to that lock. This still leaves the lock referesh issue unresolved.

julian.reschke@greenbytes.de (2004-06-02): Re.: using Lock-Token to
identify the lock to be refreshed
(http://lists.w3.org/Archives/Public/w3c-dist-auth/2004AprJun/0127.html):
problems with current rfc2518bis-05 wording; also no support in
popular implementations; suggestion to roll-back changes in -bis and
keep "If" header based syntax.

Resolution (2004-07-10): Do not change existing RFC2518 semantics
now. See discussion around

F.3 088_DAVOWNER_FIELD_IS_CLIENT_CONTROLED

Type: change

(): The DAV:owner field of a lock is controlled by the locking client
and should not be manipulated by the server. This is the only place
the client can store info. The roundtrip details should match what
we resolve for the PROP_ROUNDTRIP issue. Examples should also be
checked.

Resolution (2004-07-03): Resolved by repeated statement and no
disagreement.

Appendix G. Open issues (to be removed by RFC Editor prior to
publication)

G.1 import-gulp

Type: change

julian.reschke@greenbytes.de (2004-05-25): Make specification text
compatible with GULP where it isn’t. Integrate GULP as normative
specification of the locking behaviour.

G.2 edit

Type: edit

julian.reschke@greenbytes.de (2004-05-25): Umbrella issue for
editorial fixes/enhancements.

G.3 099_COPYMOVE_LOCKED_STATUS_CODE_CLARIFICATION
Type: change

<http://lists.w3.org/Archives/Public/w3c-dist-auth/2002AprJun/0079.html>
ccjason@us.ibm.com (): What resource should be flagged in the multistatus response to locking issues in COPY/MOVE requests?

Resolution: Resolved to flag the locking errors at the source resource that was affected by the problem. The details of how to describe the error was deferred to a subsequent version of WebDAV. - 6/15/02 - 2518bis does not reflect this.

G.4 100_COPYMOVE_LOCKED_STATUS_DESCRIPTION

Type: change

<http://lists.w3.org/Archives/Public/w3c-dist-auth/2002AprJun/0079.html>

(): The method of describing the details of (beyond what resolved by COPYMOVE_LOCKED_STATUS_CODE_CLARIFICATION) of the underlying cause of various locking and ACL COPY/MOVE problems is deferred. Two proposals were outlined in the discussion, but interest was not great and we clearly don’t have interoperability to take these proposals forward.

G.5 066_MUST_AN_IF_HEADER_CHECK_THE_ROOT_OF_URL

Type: change

(): Right now the server uses the IF: header to verify that a client knows what locks it has that are affected by an operation before it allows the operation. Must the client provide the root URL of a lock, any URL for a pertinent loc, or some specific URL in the IF: header.

ccjason@us.ibm.com (): It is felt by the group that it’s important that the client not just own and hold the lock token, but that it also know where the lock is rooted before it does tasks related to that lock. This is just a point of info. The issue itself still needs to be brought up and answered.

julian.reschke@greenbytes.de (): Summary: current implementations do not seem to care (see http://lists.w3.org/Archives/Public/w3c-dist-auth/2004AprJun/0190.html). Suggestion to require clients to specify the lock root anyway, because this is what the WG agreed upon earlier.
Index

4
423 Locked (status code) 26

C
Condition Names
DAV:create-lock (post) 19
DAV:create-resource (post) 19
DAV:lock-removal-allowed (pre) 25
DAV:lock-removed (post) 26
DAV:lock-submission-allowed (pre) 23
DAV:lock-token-matches (pre) 25
DAV:lock-token-submission-allowed (pre) 27
DAV:locks-refreshed (post) 23
DAV:name-allowed (pre) 27
DAV:need-lock-token (pre) 27
DAV:parent-resource-must-be-non-null (pre) 27

D
DAV header
compliance class ’2’ 29
compliance class ’locking’ 29
DAV:create-lock postcondition 19
DAV:create-resource postcondition 19
DAV:lock-removal-allowed precondition 25
DAV:lock-removed postcondition 26
DAV:lock-submission-allowed precondition 23
DAV:lock-token-matches precondition 25
DAV:lock-token-submission-allowed precondition 27
DAV:lockdiscovery property 15
DAV:locks-refreshed postcondition 23
DAV:name-allowed precondition 27
DAV:need-lock-token precondition 27
DAV:parent-resource-must-be-non-null precondition 27
DAV:supportedlock property 17

H
Headers
Lock-Token 28
Timeout 29

L
LOCK method 18
lock creation 18
lock refresh 23
Lock-Token header 28
Methods

LOCK (lock creation) 18
LOCK (lock refresh) 23
LOCK 18
UNLOCK 25

opaquelocktoken (URI scheme) 35

Properties

DAV:lockdiscovery 15
DAV:supportedlock 17

Status Codes

423 Locked 26

Timeout header 29

UNLOCK method 25
URI schemes
opaquelocktoken 35
Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

This document and the information contained herein are provided on an "AS IS" basis and THE CONTRIBUTOR, THE ORGANIZATION HE/SHE REPRESENTS OR IS SPONSORED BY (IF ANY), THE INTERNET SOCIETY AND THE INTERNET ENGINEERING TASK FORCE DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY WARRANTY THAT THE USE OF THE INFORMATION HEREIN WILL NOT INFRINGE ANY RIGHTS OR ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Copyright Statement

Copyright (C) The Internet Society (2004). This document is subject to the rights, licenses and restrictions contained in BCP 78, and except as set forth therein, the authors retain all their rights.

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.