

Network Working Group
Request for Comments: 3253
Category: Standards Track

G. Clemm
Rational Software
J. Amsden
T. Ellison
IBM
C. Kaler
Microsoft
J. Whitehead
U.C. Santa Cruz
March 2002

Versioning Extensions to WebDAV
(Web Distributed Authoring and Versioning)

Status of this Memo

This document specifies an Internet standards track protocol for the Internet community, and requests discussion and suggestions for improvements. Please refer to the current edition of the "Internet Official Protocol Standards" (STD 1) for the standardization state and status of this protocol. Distribution of this memo is unlimited.

Copyright Notice

Copyright (C) The Internet Society (2002). All Rights Reserved.

Abstract

This document specifies a set of methods, headers, and resource types that define the WebDAV (Web Distributed Authoring and Versioning) versioning extensions to the HTTP/1.1 protocol. WebDAV versioning will minimize the complexity of clients that are capable of interoperating with a variety of versioning repository managers, to facilitate widespread deployment of applications capable of utilizing the WebDAV Versioning services. WebDAV versioning includes automatic versioning for versioning-unaware clients, version history management, workspace management, baseline management, activity management, and URL namespace versioning.

Table of Contents

1 Introduction.....	6
1.1 Relationship to WebDAV.....	7
1.2 Notational Conventions.....	8
1.3 Terms.....	8
1.4 Property Values.....	11
1.4.1 Initial Property Value.....	11

1.4.2 Protected Property Value.....	12
1.4.3 Computed Property Value.....	12
1.4.4 Boolean Property Value.....	12
1.4.5 DAV:href Property Value.....	12
1.5 DAV Namespace XML Elements.....	12
1.6 Method Preconditions and Postconditions.....	12
1.6.1 Example - CHECKOUT request.....	13
1.7 Clarification of COPY Semantics with Overwrite:T.....	13
1.8 Versioning Methods and Write Locks.....	14
2 Basic Versioning Features.....	14
2.1 Basic Versioning Packages.....	14
2.2 Basic Versioning Semantics.....	16
2.2.1 Creating a Version-Controlled Resource.....	16
2.2.2 Modifying a Version-Controlled Resource.....	17
2.2.3 Reporting.....	19
3 Version-Control Feature.....	20
3.1 Additional Resource Properties.....	20
3.1.1 DAV:comment.....	20
3.1.2 DAV:creator-displayname.....	20
3.1.3 DAV:supported-method-set (protected).....	20
3.1.4 DAV:supported-live-property-set (protected).....	21
3.1.5 DAV:supported-report-set (protected).....	21
3.2 Version-Controlled Resource Properties.....	21
3.2.1 DAV:checked-in (protected).....	21
3.2.2 DAV:auto-version.....	22
3.3 Checked-Out Resource Properties.....	22
3.3.1 DAV:checked-out (protected).....	23
3.3.2 DAV:predecessor-set.....	23
3.4 Version Properties.....	23
3.4.1 DAV:predecessor-set (protected).....	23
3.4.2 DAV:successor-set (computed).....	23
3.4.3 DAV:checkout-set (computed).....	23
3.4.4 DAV:version-name (protected).....	24
3.5 VERSION-CONTROL Method.....	24
3.5.1 Example - VERSION-CONTROL.....	25
3.6 REPORT Method.....	25
3.7 DAV:version-tree Report.....	26
3.7.1 Example - DAV:version-tree Report.....	27
3.8 DAV:expand-property Report.....	29
3.8.1 Example - DAV:expand-property.....	30
3.9 Additional OPTIONS Semantics.....	31
3.10 Additional PUT Semantics.....	31
3.11 Additional PROPFIND Semantics.....	32
3.12 Additional PROPPATCH Semantics.....	33
3.13 Additional DELETE Semantics.....	33
3.14 Additional COPY Semantics.....	34
3.15 Additional MOVE Semantics.....	34
3.16 Additional UNLOCK Semantics.....	35

4 Checkout-In-Place Feature.....	35
4.1 Additional Version Properties.....	35
4.1.1 DAV:checkout-fork.....	36
4.1.2 DAV:checkin-fork.....	36
4.2 Checked-Out Resource Properties.....	36
4.2.1 DAV:checkout-fork.....	36
4.2.2 DAV:checkin-fork.....	37
4.3 CHECKOUT Method.....	37
4.3.1 Example - CHECKOUT.....	38
4.4 CHECKIN Method.....	38
4.4.1 Example - CHECKIN.....	40
4.5 UNCHECKOUT Method.....	40
4.5.1 Example - UNCHECKOUT.....	41
4.6 Additional OPTIONS Semantics.....	42
5 Version-History Feature.....	42
5.1 Version History Properties.....	42
5.1.1 DAV:version-set (protected).....	42
5.1.2 DAV:root-version (computed).....	42
5.2 Additional Version-Controlled Resource Properties.....	42
5.2.1 DAV:version-history (computed).....	43
5.3 Additional Version Properties.....	43
5.3.1 DAV:version-history (computed).....	43
5.4 DAV:locate-by-history Report.....	43
5.4.1 Example - DAV:locate-by-history Report.....	44
5.5 Additional OPTIONS Semantics.....	45
5.6 Additional DELETE Semantics.....	46
5.7 Additional COPY Semantics.....	46
5.8 Additional MOVE Semantics.....	46
5.9 Additional VERSION-CONTROL Semantics.....	46
5.10 Additional CHECKIN Semantics.....	47
6 Workspace Feature.....	47
6.1 Workspace Properties.....	48
6.1.1 DAV:workspace-checkout-set (computed).....	48
6.2 Additional Resource Properties.....	48
6.2.1 DAV:workspace (protected).....	48
6.3 MKWORKSPACE Method.....	48
6.3.1 Example - MKWORKSPACE.....	49
6.4 Additional OPTIONS Semantics.....	49
6.4.1 Example - OPTIONS.....	51
6.5 Additional DELETE Semantics.....	51
6.6 Additional MOVE Semantics.....	52
6.7 Additional VERSION-CONTROL Semantics.....	52
6.7.1 Example - VERSION-CONTROL.....	53
7 Update Feature.....	53
7.1 UPDATE Method.....	53
7.1.1 Example - UPDATE.....	55
7.2 Additional OPTIONS Semantics.....	55
8 Label Feature.....	56

8.1 Additional Version Properties.....	56
8.1.1 DAV:label-name-set (protected).....	56
8.2 LABEL Method.....	56
8.2.1 Example - Setting a label.....	58
8.3 Label Header.....	58
8.4 Additional OPTIONS Semantics.....	59
8.5 Additional GET Semantics.....	59
8.6 Additional PROPFIND Semantics.....	59
8.7 Additional COPY Semantics.....	60
8.8 Additional CHECKOUT Semantics.....	60
8.9 Additional UPDATE Semantics.....	61
9 Working-Resource Feature.....	62
9.1 Additional Version Properties.....	62
9.1.1 DAV:checkout-fork.....	62
9.1.2 DAV:checkin-fork.....	63
9.2 Working Resource Properties.....	63
9.2.1 DAV:auto-update (protected).....	63
9.2.2 DAV:checkout-fork.....	63
9.2.3 DAV:checkin-fork.....	63
9.3 CHECKOUT Method (applied to a version).....	63
9.3.1 Example - CHECKOUT of a version.....	65
9.4 CHECKIN Method (applied to a working resource).....	65
9.4.1 Example - CHECKIN of a working resource.....	66
9.5 Additional OPTIONS Semantics.....	67
9.6 Additional COPY Semantics.....	67
9.7 Additional MOVE Semantics.....	67
10 Advanced Versioning Features.....	67
10.1 Advanced Versioning Packages.....	68
10.2 Advanced Versioning Terms.....	68
11 MERGE Feature.....	70
11.1 Additional Checked-Out Resource Properties.....	70
11.1.1 DAV:merge-set.....	70
11.1.2 DAV:auto-merge-set.....	71
11.2 MERGE Method.....	71
11.2.1 Example - MERGE.....	74
11.3 DAV:merge-preview Report.....	75
11.3.1 Example - DAV:merge-preview Report.....	76
11.4 Additional OPTIONS Semantics.....	77
11.5 Additional DELETE Semantics.....	77
11.6 Additional CHECKIN Semantics.....	77
12 Baseline Feature.....	77
12.1 Version-Controlled Configuration Properties.....	78
12.1.1 DAV:baseline-controlled-collection (protected).....	78
12.2 Checked-Out Configuration Properties.....	78
12.2.1 DAV:subbaseline-set.....	78
12.3 Baseline Properties.....	78
12.3.1 DAV:baseline-collection (protected).....	79
12.3.2 DAV:subbaseline-set (protected).....	79

12.4 Additional Resource Properties.....	79
12.4.1 DAV:version-controlled-configuration (computed).....	79
12.5 Additional Workspace Properties.....	80
12.5.1 DAV:baseline-controlled-collection-set (computed).....	80
12.6 BASELINE-CONTROL Method.....	80
12.6.1 Example - BASELINE-CONTROL.....	82
12.7 DAV:compare-baseline Report.....	84
12.7.1 Example - DAV:compare-baseline Report.....	85
12.8 Additional OPTIONS Semantics.....	86
12.9 Additional MKCOL Semantics.....	86
12.10 Additional COPY Semantics.....	86
12.11 Additional CHECKOUT Semantics.....	86
12.12 Additional CHECKIN Semantics.....	86
12.13 Additional UPDATE Semantics.....	87
12.14 Additional MERGE Semantics.....	89
13 Activity Feature.....	90
13.1 Activity Properties.....	91
13.1.1 DAV:activity-version-set (computed).....	91
13.1.2 DAV:activity-checkout-set (computed).....	92
13.1.3 DAV:subactivity-set.....	92
13.1.4 DAV:current-workspace-set (computed).....	92
13.2 Additional Version Properties.....	92
13.2.1 DAV:activity-set.....	93
13.3 Additional Checked-Out Resource Properties.....	93
13.3.1 DAV:unreserved.....	93
13.3.2 DAV:activity-set.....	93
13.4 Additional Workspace Properties.....	93
13.4.1 DAV:current-activity-set.....	94
13.5 MKACTIVITY Method.....	94
13.5.1 Example - MKACTIVITY.....	95
13.6 DAV:latest-activity-version Report.....	95
13.7 Additional OPTIONS Semantics.....	96
13.8 Additional DELETE Semantics.....	96
13.9 Additional MOVE Semantics.....	97
13.10 Additional CHECKOUT Semantics.....	97
13.10.1 Example - CHECKOUT with an activity.....	98
13.11 Additional CHECKIN Semantics.....	99
13.12 Additional MERGE Semantics.....	99
14 Version-Controlled-Collection Feature.....	100
14.1 Version-Controlled Collection Properties.....	102
14.1.1 DAV:eclipsed-set (computed).....	102
14.2 Collection Version Properties.....	103
14.2.1 DAV:version-controlled-binding-set (protected).....	103
14.3 Additional OPTIONS Semantics.....	103
14.4 Additional DELETE Semantics.....	103
14.5 Additional MKCOL Semantics.....	104
14.6 Additional COPY Semantics.....	104
14.7 Additional MOVE Semantics.....	104

14.8	Additional VERSION-CONTROL Semantics.....	104
14.9	Additional CHECKOUT Semantics.....	105
14.10	Additional CHECKIN Semantics.....	105
14.11	Additional UPDATE and MERGE Semantics.....	106
15	Internationalization Considerations.....	106
16	Security Considerations.....	107
16.1	Auditing and Traceability.....	107
16.2	Increased Need for Access Control.....	108
16.3	Security Through Obscurity.....	108
16.4	Denial of Service.....	108
17	IANA Considerations.....	109
18	Intellectual Property.....	109
19	Acknowledgements.....	109
20	References.....	110
	Appendix A - Resource Classification.....	111
A.1	DeltaV-Compliant Unmapped URL.....	111
A.2	DeltaV-Compliant Resource.....	111
A.3	DeltaV-Compliant Collection.....	112
A.4	Versionable Resource.....	112
A.5	Version-Controlled Resource.....	112
A.6	Version.....	113
A.7	Checked-In Version-Controlled Resource.....	113
A.8	Checked-Out Resource.....	113
A.9	Checked-Out Version-Controlled Resource.....	114
A.10	Working Resource.....	114
A.11	Version History.....	114
A.12	Workspace.....	115
A.13	Activity.....	115
A.14	Version-Controlled Collection.....	115
A.15	Collection Version.....	115
A.16	Version-Controlled Configuration.....	116
A.17	Baseline.....	116
A.18	Checked-Out Version-Controlled Configuration.....	116
	Authors' Addresses.....	117
	Full Copyright Statement.....	118

1 Introduction

This document specifies a set of methods, headers, and properties that define the WebDAV (Web Distributed Authoring and Versioning) versioning extensions to the HTTP/1.1 protocol. Versioning is concerned with tracking and accessing the history of important states of a web resource, such as a standalone web page. The benefits of versioning in the context of the worldwide web include:

- A resource has an explicit history and a persistent identity across the various states it has had during the course of that history. It allows browsing through past and alternative versions of a resource. Frequently the modification and authorship history of a resource is critical information in itself.
- Resource states (versions) are given stable names that can support externally stored links for annotation and link server support. Both annotation and link servers frequently need to store stable references to portions of resources that are not under their direct control. By providing stable states of resources, version control systems allow not only stable pointers into those resources, but also well defined methods to determine the relationships of those states of a resource.

WebDAV Versioning defines both basic and advanced versioning functionality.

Basic versioning allows users to:

- Put a resource under version control
- Determine whether a resource is under version control
- Determine whether a resource update will automatically be captured as a new version
- Create and access distinct versions of a resource

Advanced versioning provides additional functionality for parallel development and configuration management of sets of web resources.

This document will first define the properties and method semantics for the basic versioning features, and then define the additional properties and method semantics for the advanced versioning features. An implementer that is only interested in basic versioning should skip the advanced versioning sections (Section 10 to Section 14).

1.1 Relationship to WebDAV

To maximize interoperability and the use of existing protocol functionality, versioning support is designed as extensions to the WebDAV protocol [RFC2518], which itself is an extension to the HTTP protocol [RFC2616]. All method marshalling and postconditions defined by RFC 2518 and RFC 2616 continue to hold, to ensure that versioning unaware clients can interoperate successfully with versioning servers. Although the versioning extensions are designed to be orthogonal to most aspects of the WebDAV and HTTP protocols, a clarification to RFC 2518 is required for effective interoperable versioning. This clarification is described in Section 1.7.

1.2 Notational Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

The term "protected" is placed in parentheses following the definition of a protected property (see Section 1.4.2).

The term "computed" is placed in parentheses following the definition of a computed property (see Section 1.4.3).

When an XML element type in the "DAV:" namespace is referenced in this document outside of the context of an XML fragment, the string "DAV:" will be prefixed to the element type.

When a method is defined in this document, a list of preconditions and postconditions will be defined for that method. If the semantics of an existing method is being extended, a list of additional preconditions and postconditions will be defined. A precondition or postcondition is prefixed by a parenthesized XML element type that identifies that precondition or postcondition (see Section 1.6).

1.3 Terms

This document uses the terms defined in RFC 2616, in RFC 2518, and in this section. Section 2.2 defines the semantic versioning model underlying this terminology.

Version Control, Checked-In, Checked-Out

"Version control" is a set of constraints on how a resource can be updated. A resource under version control is either in a "checked-in" or "checked-out" state, and the version control constraints apply only while the resource is in the checked-in state.

Versionable Resource

A "versionable resource" is a resource that can be put under version control.

Version-Controlled Resource

When a versionable resource is put under version control, it becomes a "version-controlled resource". A version-controlled resource can be "checked out" to allow modification of its content or dead properties by standard HTTP and WebDAV methods.

Checked-Out Resource

A "checked-out resource" is a resource under version control that is in the checked-out state.

Version Resource

A "version resource", or simply "version", is a resource that contains a copy of a particular state (content and dead properties) of a version-controlled resource. A version is created by "checking in" a checked-out resource. The server allocates a distinct new URL for each new version, and this URL will never be used to identify any resource other than that version. The content and dead properties of a version never change.

Version History Resource

A "version history resource", or simply "version history", is a resource that contains all the versions of a particular version-controlled resource.

Version Name

A "version name" is a string chosen by the server to distinguish one version of a version history from the other versions of that version history. Versions from different version histories may have the same version name.

Predecessor, Successor, Ancestor, Descendant

When a version-controlled resource is checked out and then subsequently checked in, the version that was checked out becomes a "predecessor" of the version created by the checkin. A client can specify multiple predecessors for a new version if the new version is logically a merge of those predecessors. When a version is connected to another version by traversing one or more predecessor relations, it is called an "ancestor" of that version. The inverse of the predecessor and ancestor relations are the "successor" and "descendant" relations. Therefore, if X is a predecessor of Y, then Y is a successor of X, and if X is an ancestor of Y, then Y is a descendant of X.

Root Version Resource

The "root version resource", or simply "root version", is the version in a version history that is an ancestor of every other version in that version history.

Workspace Resource

A "workspace resource", or simply "workspace", is a collection that contains at most one version-controlled resource for a given version history (see Section 6).

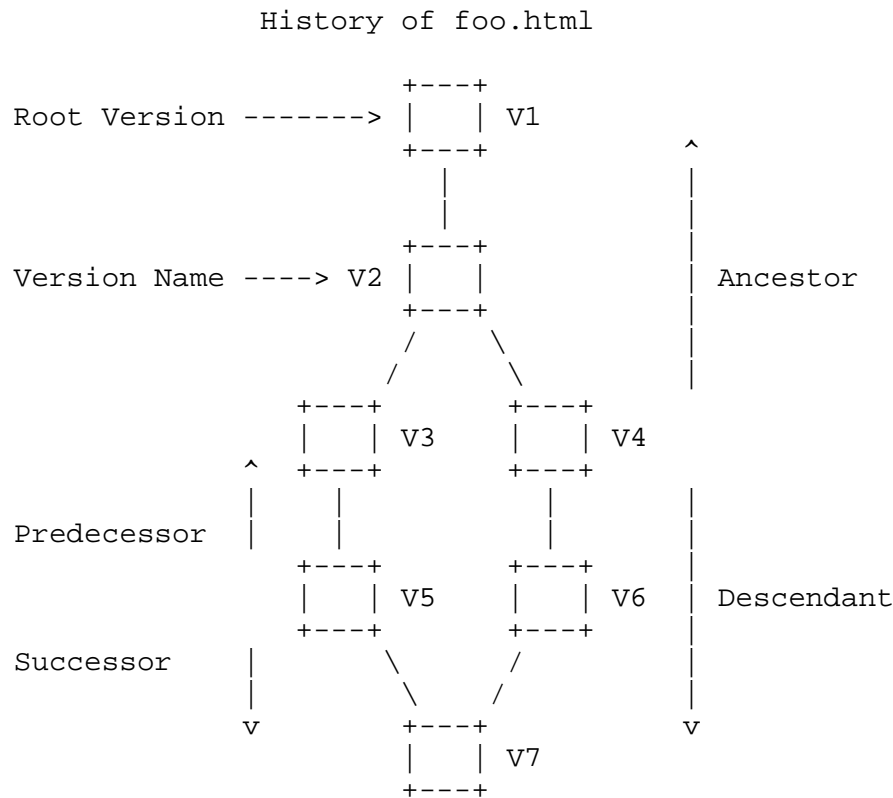
Working Resource

A "working resource" is a checked-out resource created by the server at a server-defined URL when a version (instead of a version-controlled resource) is checked out. Unlike a checked-out version-controlled resource, a working resource is deleted when it is checked in.

Fork, Merge

When a second successor is added to a version, this creates a "fork" in the version history. When a version is created with multiple predecessors, this creates a "merge" in the version history. A server may restrict the version history to be linear (with no forks or merges), but an interoperable versioning client should be prepared to deal with both forks and merges in the version history.

The following diagram illustrates several of the previous definitions. Each box represents a version and each line between two boxes represents a predecessor/successor relationship. For example, it shows V3 is a predecessor of V5, V7 is a successor of V5, V1 is an ancestor of V4, and V7 is a descendant of V4. It also shows that there is a fork at version V2 and a merge at version V7.



Label

A "label" is a name that can be used to select a version from a version history. A label can be assigned by either a client or the server. The same label can be used in different version histories.

1.4 Property Values

1.4.1 Initial Property Value

Unless an initial value of a property of a given type is defined by this document, the initial value of a property of that type is implementation dependent.

1.4.2 Protected Property Value

When a property of a specific kind of resource is "protected", the property value cannot be updated on that kind of resource except by a method explicitly defined as updating that specific property. In particular, a protected property cannot be updated with a PROPPATCH request. Note that a given property can be protected on one kind of resource, but not protected on another kind of resource.

1.4.3 Computed Property Value

When a property is "computed", its value is defined in terms of a computation based on the content and other properties of that resource, or even of some other resource. When the semantics of a method is defined in this document, the effect of that method on non-computed properties will be specified; the effect of that method on computed properties will not be specified, but can be inferred from the computation defined for those properties. A computed property is always a protected property.

1.4.4 Boolean Property Value

Some properties take a Boolean value of either "false" or "true".

1.4.5 DAV:href Property Value

The DAV:href XML element is defined in RFC 2518, Section 12.3.

1.5 DAV Namespace XML Elements in Request and Response Bodies

Although WebDAV request and response bodies can be extended by arbitrary XML elements, which can be ignored by the message recipient, an XML element in the DAV namespace MUST NOT be used in the request or response body of a versioning method unless that XML element is explicitly defined in an IETF RFC.

1.6 Method Preconditions and Postconditions

A "precondition" of a method describes the state of the server that must be true for that method to be performed. A "postcondition" of a method describes the state of the server that must be true after that method has been completed. If a method precondition or postcondition for a request is not satisfied, 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 403 and 409 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:responsedescription element.

1.6.1 Example - CHECKOUT request with DAV:must-be-checked-in response

>>REQUEST

```
CHECKOUT /foo.html HTTP/1.1
Host: www.webdav.org
```

>>RESPONSE

```
HTTP/1.1 409 Conflict
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:error xmlns:D="DAV:">
  <D:must-be-checked-in/>
</D:error>
```

In this example, the request to CHECKOUT /foo.html fails because /foo.html is not checked in.

1.7 Clarification of COPY Semantics with Overwrite:T

RFC 2518, Section 8.8.4 states:

"If a resource exists at the destination and the Overwrite header is "T" then prior to performing the copy the server MUST perform a DELETE with "Depth: infinity" on the destination resource."

The purpose of this sentence is to ensure that following a COPY, all destination resources have the same content and dead properties as the corresponding resources identified by the request-URL (where a resource with a given name relative to the Destination URL "corresponds" to a resource with the same name relative to the request-URL). If at the time of the request, there already is a resource at the destination that has the same resource type as the corresponding resource at the request-URL, that resource MUST NOT be deleted, but MUST be updated to have the content and dead properties

of its corresponding member. If a client wishes all resources at the destination to be deleted prior to the COPY, it MUST explicitly issue a DELETE request.

The difference between updating a resource and replacing a resource with a new resource is especially important when resource history is being maintained (the former adds to an existing history, while the latter creates a new history). In addition, locking and access control constraints might allow you to update a resource, but not allow you to delete it and create a new one in its place.

Note that this clarification does not apply to a MOVE request. A MOVE request with Overwrite:T MUST perform the DELETE with "Depth:infinity" on the destination resource prior to performing the MOVE.

1.8 Versioning Methods and Write Locks

If a write-locked resource has a non-computed property defined by this document, the property value MUST NOT be changed by a request unless the appropriate lock token is included in the request. Since every method introduced in this document other than REPORT modifies at least one property defined by this document, every versioning method other than REPORT is affected by a write lock. In particular, the method MUST fail with a 423 (Locked) status if the resource is write-locked and the appropriate token is not specified in an If request header.

2 Basic Versioning Features

Each basic versioning feature defines extensions to existing HTTP and WebDAV methods, as well as new resource types, live properties, and methods.

2.1 Basic Versioning Packages

A server MAY support any combination of versioning features. However, in order to minimize the complexity of a WebDAV basic versioning client, a WebDAV basic versioning server SHOULD support one of the following three "packages" (feature sets):

- Core-Versioning Package: version-control
- Basic-Server-Workspace Package: version-control, workspace, version-history, checkout
- Basic-Client-Workspace Package: version-control, working-resource, update, label

The core-versioning package supports linear versioning by both versioning-aware and versioning-unaware clients. A versioning-aware client can use reports and properties to access previous versions of a version-controlled resource.

The basic workspace packages support parallel development of version-controlled resources. Each client has its own configuration of the shared version-controlled resources, and can make changes to its configuration without disturbing that of another client.

In the basic-server-workspace package, all persistent state is maintained on the server. Each client has its own workspace resource allocated on the server, where each workspace identifies a configuration of the shared version-controlled resources. Each client makes changes to its workspace, and can transfer changes when appropriate from one workspace to another. The server workspace package is appropriate for clients with no local persistent state, or for clients that wish to expose their working configurations to other clients.

In the basic-client-workspace package, each client maintains in local persistent storage the state for its configuration of the shared version-controlled resources. When a client is ready to make its changes visible to other clients, it allocates a set of "working resources" on the server, updates the content and dead properties of these working resources, and then uses the set of working resources to update the version-controlled resources. The working resources are used, instead of directly updating the version-controlled resources, so that sets of consistent updates can be prepared in parallel by multiple clients. Also, a working resource allows a client to prepare a single update that requires multiple server requests (e.g. updating both the content and dead properties of a resource requires both a PUT and a PROPPATCH). The client workspace package simplifies the server implementation by requiring each client to maintain its own namespace, but this requires that the clients have local persistent state, and does not allow clients to expose their working configurations to other clients.

A server that supports both basic workspace packages will interoperate with all basic versioning clients.

2.2 Basic Versioning Semantics

2.2.1 Creating a Version-Controlled Resource

In order to track the history of the content and dead properties of a versionable resource, a user can put the resource under version control with a VERSION-CONTROL request. A VERSION-CONTROL request performs three distinct operations:

- 1) It creates a new "version history resource". In basic versioning, a version history resource is not assigned a URL, and hence is not visible in the http scheme URL space. However, when the version-history feature (see Section 5) is supported, this changes, and each version history resource is assigned a new distinct and unique server-defined URL.
- 2) It creates a new "version resource" and adds it to the new version history resource. The body and dead properties of the new version resource are a copy of those of the versionable resource.

The server assigns the new version resource a new distinct and unique URL.

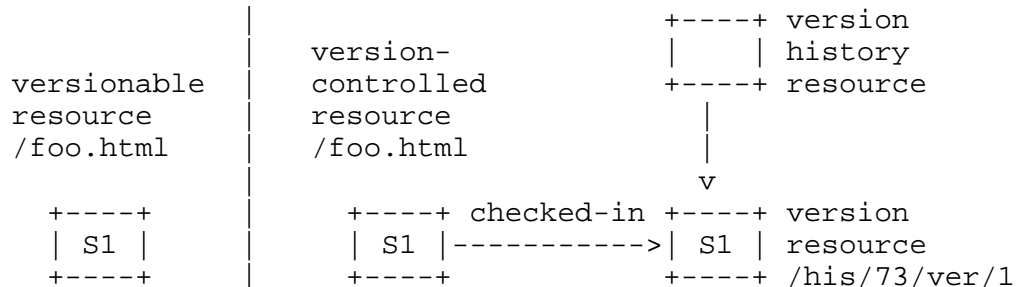
- 3) It converts the versionable resource into a "version-controlled resource". The version-controlled resource continues to be identified by the same URL that identified it as a versionable resource. As part of this conversion, it adds a DAV:checked-in property, whose value contains the URL of the new version resource.

Note that a versionable resource and a version-controlled resource are not new types of resources (i.e. they introduce no new DAV:resourcetype), but rather they are any type of resource that supports the methods and live properties defined for them in this document, in addition to all the methods and live properties implied by their DAV:resourcetype. For example, a collection (whose DAV:resourcetype is DAV:collection) is a versionable resource if it supports the VERSION-CONTROL method, and is a version-controlled resource if it supports the version-controlled resource methods and live properties.

In the following example, foo.html is a versionable resource that is put under version control. After the VERSION-CONTROL request succeeds, there are two additional resources: a new version history resource and a new version resource in that version history. The versionable resource is converted into a version-controlled resource, whose DAV:checked-in property identifies the new version resource.

The content and dead properties of a resource are represented by the symbol appearing inside the box for that resource (e.g., "S1" in the following example).

===VERSION-CONTROL==>



Thus, whereas before the VERSION-CONTROL request there was only one, non-version-controlled resource, after VERSION-CONTROL there are three separate, distinct resources, each containing its own state and properties: the version-controlled resource, the version resource, and the version history resource. Since the version-controlled resource and the version resource are separate, distinct resources, when a method is applied to a version-controlled resource, it is only applied to that version-controlled resource, and is not applied to the version resource that is currently identified by the DAV:checked-in property of that version-controlled resource. Although the content and dead properties of a checked-in version-controlled resource are required to be the same as those of its current DAV:checked-in version, its live properties may differ. An implementation may optimize storage by retrieving the content and dead properties of a checked-in version-controlled resource from its current DAV:checked-in version rather than storing them in the version-controlled resource, but this is just an implementation optimization.

Normally, a resource is placed under version control with an explicit VERSION-CONTROL request. A server MAY automatically place every new versionable resource under version control. In this case, the resulting state on the server MUST be the same as if the client had explicitly applied a VERSION-CONTROL request to the versionable resource.

2.2.2 Modifying a Version-Controlled Resource

In order to use methods like PUT and PROPPATCH to directly modify the content or dead properties of a version-controlled resource, the version-controlled resource must first be checked out. When the checked-out resource is checked in, a new version is created in the

version history of that version-controlled resource. The version that was checked out is remembered as the predecessor of the new version.

The DAV:auto-version property (see Sections 3.2.2) of a checked-in version-controlled resource determines how it responds to a method that attempts to modify its content or dead properties. Possible responses include:

- Fail the request. The resource requires an explicit CHECKOUT request for it to be modified (see Sections 4 and 9.2.1).
- Automatically checkout the resource, perform the modification, and automatically checkin the resource. This ensures that every state of the resource is tracked by the server, but can result in an excessive number of versions being created.
- Automatically checkout the resource, perform the modification, and then if the resource is not write-locked, automatically checkin the resource. If the resource is write-locked, it remains checked-out until the write-lock is removed (either explicitly through a subsequent UNLOCK request or implicitly through a time-out of the write-lock). This helps a locking client avoid the proliferation of versions, while still allowing a non-locking client to update the resource.
- Automatically checkout the resource, perform the modification, and then leave the resource checked out. If the resource is write-locked, it will be automatically checked in when the write-lock is removed, but an explicit CHECKIN operation (see Section 4.4) is required for a non-write-locked resource. This minimizes the number of new versions that will be created by a versioning unaware client, but only a versioning aware client can create new versions of a non-write-locked resource.
- Fail the request unless the resource is write-locked. If it is write-locked, automatically checkout the resource and perform the modification. The resource is automatically checked in when the write-lock is removed. This minimizes the number of new versions that will be created by a versioning unaware client, but never automatically checks out a resource that will not subsequently be automatically checked in.

The following diagram illustrates the effect of the checkout/checkin process on a version-controlled resource and its version history. The symbol inside a box (S1, S2, S3) represents the current content and dead properties of the resource represented by that box. The symbol next to a box (V1, V2, V3) represents the URL for that resource.

```

===checkout==>      ===PUT==>      ===checkin==>

/foo.html (version-controlled resource)

+----+      +----+      +----+      +----+
| S2 |      | S2 |      | S3 |      | S3 |
+----+      +----+      +----+      +----+
Checked-In=V2 | Checked-Out=V2 | Checked-Out=V2 | Checked-In=V3

/his/73 (version history for /foo.html)

+----+      +----+      +----+      +----+
| S1 | V1    | S1 | V1    | S1 | V1    | S1 | V1
+----+      +----+      +----+      +----+
|      |      |      |      |      |      |
+----+      +----+      +----+      +----+
| S2 | V2    | S2 | V2    | S2 | V2    | S2 | V2
+----+      +----+      +----+      +----+
|      |      |      |      |      |      |
+----+      +----+      +----+      +----+
| S3 | V3    | S3 | V3    | S3 | V3    | S3 | V3
+----+      +----+      +----+      +----+

```

Note that a version captures only a defined subset of the state of a resource. In particular, a version of a basic resource captures its content and dead properties, but not its live properties.

2.2.3 Reporting

Some versioning information about a resource requires that parameters be specified along with that request for information. Included in basic versioning is the required support for an extensible reporting mechanism, which includes a REPORT method as well as a live property for determining what reports are supported by a particular resource. The REPORT method is required by versioning, but it can be used in non-versioning WebDAV extensions.

To allow a client to query the properties of all versions in the version history of a specified version-controlled resource, basic versioning provides the DAV:version-tree report (see Section 3.7). A more powerful version history reporting mechanism is provided by applying the DAV:expand-property report (see Section 3.8) to a version history resource (see Section 5).

3 Version-Control Feature

The version-control feature provides support for putting a resource under version control, creating an associated version-controlled resource and version history resource as described in Section 2.2.1. A server indicates that it supports the version-control feature by including the string "version-control" as a field in the DAV header in the response to an OPTIONS request. The version-control feature MUST be supported if any other versioning feature is supported.

3.1 Additional Resource Properties

The version-control feature introduces the following REQUIRED properties for any WebDAV resource.

3.1.1 DAV:comment

This property is used to track a brief comment about a resource that is suitable for presentation to a user. The DAV:comment of a version can be used to indicate why that version was created.

```
<!ELEMENT comment (#PCDATA)>
PCDATA value: string
```

3.1.2 DAV:creator-displayname

This property contains a description of the creator of the resource that is suitable for presentation to a user. The DAV:creator-displayname of a version can be used to indicate who created that version.

```
<!ELEMENT creator-displayname (#PCDATA)>
PCDATA value: string
```

3.1.3 DAV:supported-method-set (protected)

This property identifies the methods that are supported by the resource. A method is supported by a resource if there is some state of that resource for which an application of that method will

successfully satisfy all postconditions of that method, including any additional postconditions added by the features supported by that resource.

```
<!ELEMENT supported-method-set (supported-method*)>
<!ELEMENT supported-method ANY>
<!ATTLIST supported-method name NMTOKEN #REQUIRED>
name value: a method name
```

3.1.4 DAV:supported-live-property-set (protected)

This property identifies the live properties that are supported by the resource. A live property is supported by a resource if that property has the semantics defined for that property. The value of this property MUST identify all live properties defined by this document that are supported by the resource, and SHOULD identify all live properties that are supported by the resource.

```
<!ELEMENT supported-live-property-set (supported-live-property*)>
<!ELEMENT supported-live-property name>
<!ELEMENT prop ANY>
ANY value: a property element type
```

3.1.5 DAV:supported-report-set (protected)

This property identifies the reports that are supported by the resource.

```
<!ELEMENT supported-report-set (supported-report*)>
<!ELEMENT supported-report report>
<!ELEMENT report ANY>
ANY value: a report element type
```

3.2 Version-Controlled Resource Properties

The version-control feature introduces the following REQUIRED properties for a version-controlled resource.

3.2.1 DAV:checked-in (protected)

This property appears on a checked-in version-controlled resource, and identifies a version that has the same content and dead properties as the version-controlled resource. This property is removed when the resource is checked out, and then added back (identifying a new version) when the resource is checked back in.

```
<!ELEMENT checked-in (href)>
```

3.2.2 DAV:auto-version

If the DAV:auto-version value is DAV:checkout-checkin, when a modification request (such as PUT/PROPPATCH) is applied to a checked-in version-controlled resource, the request is automatically preceded by a checkout and followed by a checkin operation.

If the DAV:auto-version value is DAV:checkout-unlocked-checkin, when a modification request is applied to a checked-in version-controlled resource, the request is automatically preceded by a checkout operation. If the resource is not write-locked, the request is automatically followed by a checkin operation.

If the DAV:auto-version value is DAV:checkout, when a modification request is applied to a checked-in version-controlled resource, the request is automatically preceded by a checkout operation.

If the DAV:auto-version value is DAV:locked-checkout, when a modification request is applied to a write-locked checked-in version-controlled resource, the request is automatically preceded by a checkout operation.

If an update to a write-locked checked-in resource is automatically preceded by a checkout of that resource, the checkout is associated with the write lock. When this write lock is removed (e.g. from an UNLOCK or a lock timeout), if the resource has not yet been checked in, the removal of the write lock is automatically preceded by a checkin operation.

A server MAY refuse to allow the value of the DAV:auto-version property to be modified, or MAY only support values from a subset of the valid values.

```
<!ELEMENT auto-version (checkout-checkin | checkout-unlocked-checkin
| checkout | locked-checkout)? >
<!ELEMENT checkout-checkin EMPTY>
<!ELEMENT checkout-unlocked-checkin EMPTY>
<!ELEMENT checkout EMPTY>
<!ELEMENT locked-checkout EMPTY>
```

3.3 Checked-Out Resource Properties

The version-control feature introduces the following REQUIRED properties for a checked-out resource.

3.3.1 DAV:checked-out (protected)

This property identifies the version that was identified by the DAV:checked-in property at the time the resource was checked out. This property is removed when the resource is checked in.

```
<!ELEMENT checked-out (href)>
```

3.3.2 DAV:predecessor-set

This property determines the DAV:predecessor-set property of the version that results from checking in this resource.

A server MAY reject attempts to modify the DAV:predecessor-set of a version-controlled resource.

```
<!ELEMENT predecessor-set (href+)>
```

3.4 Version Properties

The version-control feature introduces the following REQUIRED properties for a version.

3.4.1 DAV:predecessor-set (protected)

This property identifies each predecessor of this version. Except for the root version, which has no predecessors, each version has at least one predecessor.

```
<!ELEMENT predecessor-set (href*)>
```

3.4.2 DAV:successor-set (computed)

This property identifies each version whose DAV:predecessor-set identifies this version.

```
<!ELEMENT successor-set (href*)>
```

3.4.3 DAV:checkout-set (computed)

This property identifies each checked-out resource whose DAV:checked-out property identifies this version.

```
<!ELEMENT checkout-set (href*)>
```

3.4.4 DAV:version-name (protected)

This property contains a server-defined string that is different for each version in a given version history. This string is intended for display for a user, unlike the URL of a version, which is normally only used by a client and not displayed for a user.

```
<!ELEMENT version-name (#PCDATA)>
PCDATA value: string
```

3.5 VERSION-CONTROL Method

A VERSION-CONTROL request can be used to create a version-controlled resource at the request-URL. It can be applied to a versionable resource or to a version-controlled resource.

If the request-URL identifies a versionable resource, a new version history resource is created, a new version is created whose content and dead properties are copied from the versionable resource, and the resource is given a DAV:checked-in property that is initialized to identify this new version.

If the request-URL identifies a version-controlled resource, the resource just remains under version-control. This allows a client to be unaware of whether or not a server automatically puts a resource under version control when it is created.

If a VERSION-CONTROL request fails, the server state preceding the request MUST be restored.

Marshalling:

If a request body is included, it MUST be a DAV:version-control XML element.

```
<!ELEMENT version-control ANY>
```

If a response body for a successful request is included, it MUST be a DAV:version-control-response XML element. Note that this document does not define any elements for the VERSION-CONTROL response body, but the DAV:version-control-response element is defined to ensure interoperability between future extensions that do define elements for the VERSION-CONTROL response body.

```
<!ELEMENT version-control-response ANY>
```


Postconditions:

(DAV:put-under-version-control): If the request-URL identified a versionable resource at the time of the request, the request MUST have created a new version history and MUST have created a new version resource in that version history. The resource MUST have a DAV:checked-in property that identifies the new version. The content, dead properties, and DAV:resourcetype of the new version MUST be the same as those of the resource. Note that an implementation can choose to locate the version history and version resources anywhere that it wishes. In particular, it could locate them on the same host and server as the version-controlled resource, on a different virtual host maintained by the same server, on the same host maintained by a different server, or on a different host maintained by a different server.

(DAV:must-not-change-existing-checked-in-out): If the request-URL identified a resource already under version control at the time of the request, the request MUST NOT change the DAV:checked-in or DAV:checked-out property of that version-controlled resource.

3.5.1 Example - VERSION-CONTROL

>>REQUEST

```
VERSION-CONTROL /foo.html HTTP/1.1
Host: www.webdav.org
Content-Length: 0
```

>>RESPONSE

```
HTTP/1.1 200 OK
```

In this example, /foo.html is put under version control. A new version history is created for it, and a new version is created that has a copy of the content and dead properties of /foo.html. The DAV:checked-in property of /foo.html identifies this new version.

3.6 REPORT Method

A REPORT request is an extensible mechanism for obtaining information about a resource. Unlike a resource property, which has a single value, the value of a report can depend on additional information specified in the REPORT request body and in the REPORT request headers.

Marshalling:

The body of a REPORT request specifies which report is being requested, as well as any additional information that will be used to customize the report.

The request MAY include a Depth header. If no Depth header is included, Depth:0 is assumed.

The response body for a successful request MUST contain the requested report.

If a Depth request header is included, the response MUST be a 207 Multi-Status. The request MUST be applied separately to the collection itself and to all members of the collection that satisfy the Depth value. The DAV:prop element of a DAV:response for a given resource MUST contain the requested report for that resource.

Preconditions:

(DAV:supported-report): The specified report MUST be supported by the resource identified by the request-URL.

Postconditions:

(DAV:no-modification): The REPORT method MUST NOT have changed the content or dead properties of any resource.

3.7 DAV:version-tree Report

The DAV:version-tree report describes the requested properties of all the versions in the version history of a version. If the report is requested for a version-controlled resource, it is redirected to its DAV:checked-in or DAV:checked-out version.

The DAV:version-tree report MUST be supported by all version resources and all version-controlled resources.

Marshalling:

The request body MUST be a DAV:version-tree XML element.

<!ELEMENT version-tree ANY>

ANY value: a sequence of zero or more elements, with at most one DAV:prop element.

prop: see RFC 2518, Section 12.11

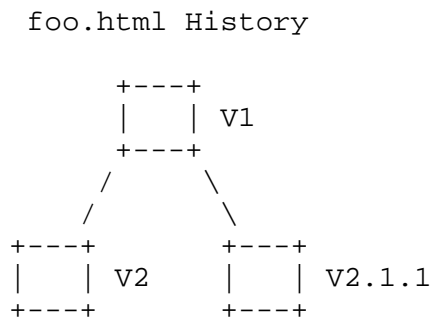
The response body for a successful request MUST be a DAV:multistatus XML element.

multistatus: see RFC 2518, Section 12.9

The response body for a successful DAV:version-tree REPORT request MUST contain a DAV:response element for each version in the version history of the version identified by the request-URL.

3.7.1 Example - DAV:version-tree Report

The version history drawn below would produce the following version tree report.



>>REQUEST

```

REPORT /foo.html HTTP/1.1
Host: www.webdav.org
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx
  
```

```

<?xml version="1.0" encoding="utf-8" ?>
<D:version-tree xmlns:D="DAV:">
  <D:prop>
    <D:version-name/>
    <D:creator-displayname/>
    <D:successor-set/>
  </D:prop>
</D:version-tree>
  
```

>>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>http://repo.webdav.org/his/23/ver/V1</D:href>

 <D:propstat>

 <D:prop>

 <D:version-name>V1</D:version-name>

 <D:creator-displayname>Fred</D:creator-displayname>

 <D:successor-set>

 <D:href>http://repo.webdav.org/his/23/ver/V2</D:href>

 <D:href>http://repo.webdav.org/his/23/ver/V2.1.1</D:href>

 </D:successor-set>

 </D:prop>

 <D:status>HTTP/1.1 200 OK</D:status>

 </D:propstat>

 </D:response>

 <D:response>

 <D:href>http://repo.webdav.org/his/23/ver/V2</D:href>

 <D:propstat>

 <D:prop>

 <D:version-name>V2</D:version-name>

 <D:creator-displayname>Fred</D:creator-displayname>

 <D:successor-set/>

 </D:prop>

 <D:status>HTTP/1.1 200 OK</D:status>

 </D:propstat>

 </D:response>

 <D:response>

 <D:href>http://repo.webdav.org/his/23/ver/V2.1.1</D:href>

 <D:propstat>

 <D:prop>

 <D:version-name>V2.1.1</D:version-name>

 <D:creator-displayname>Sally</D:creator-displayname>

 <D:successor-set/>

 </D:prop>

 <D:status>HTTP/1.1 200 OK</D:status>

 </D:propstat>

 </D:response>

</D:multistatus>

3.8 DAV:expand-property Report

Many property values are defined as a DAV:href, or a set of DAV:href elements. The DAV:expand-property report provides a mechanism for retrieving in one request the properties from the resources identified by those DAV:href elements. This report not only decreases the number of requests required, but also allows the server to minimize the number of separate read transactions required on the underlying versioning store.

The DAV:expand-property report SHOULD be supported by all resources that support the REPORT method.

Marshalling:

The request body MUST be a DAV:expand-property XML element.

```
<!ELEMENT expand-property (property*)>
<!ELEMENT property (property*)>
<!ATTLIST property name NMTOKEN #REQUIRED>
name value: a property element type
<!ATTLIST property namespace NMTOKEN "DAV:">
namespace value: an XML namespace
```

The response body for a successful request MUST be a DAV:multistatus XML element.

multistatus: see RFC 2518, Section 12.9

The properties reported in the DAV:prop elements of the DAV:multistatus element MUST be those identified by the DAV:property elements in the DAV:expand-property element. If there are DAV:property elements nested within a DAV:property element, then every DAV:href in the value of the corresponding property is replaced by a DAV:response element whose DAV:prop elements report the values of the properties identified by the nested DAV:property elements. The nested DAV:property elements can in turn contain DAV:property elements, so that multiple levels of DAV:href expansion can be requested.

Note that a validating parser MUST be aware that the DAV:expand-property report effectively modifies the DTD of every property by replacing every occurrence of "href" in the DTD with "href | response".

3.8.1 Example - DAV:expand-property

This example describes how to query a version-controlled resource to determine the DAV:creator-display-name and DAV:activity-set of every version in the version history of that version-controlled resource. This example assumes that the server supports the version-history feature (see Section 5).

>>REQUEST

```
REPORT /foo.html HTTP/1.1
Host: www.webdav.org
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:expand-property xmlns:D="DAV:">
  <D:property name="version-history">
    <D:property name="version-set">
      <D:property name="creator-displayname"/>
      <D:property name="activity-set"/>
    </D:property>
  </D:property>
</D:expand-property>
```

>>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>http://www.webdav.org/foo.html</D:href>
    <D:propstat>
      <D:prop>
        <D:version-history>
          <D:response>
            <D:href>http://repo.webdav.org/his/23</D:href>
            <D:propstat>
              <D:prop>
                <D:version-set>
                  <D:response>
                    <D:href>http://repo.webdav.org/his/23/ver/1</D:href>
                    <D:propstat>
                      <D:prop>
                        <D:creator-displayname>Fred</D:creator-displayname>
```

```

        <D:activity-set> <D:href>
            http://www.webdav.org/ws/dev/sally
        </D:href> </D:activity-set> </D:prop>
        <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat> </D:response>
<D:response>
<D:href>http://repo.webdav.org/his/23/ver/2</D:href>
    <D:propstat>
        <D:prop>
<D:creator-displayname>Sally</D:creator-displayname>
        <D:activity-set>
<D:href>http://repo.webdav.org/act/add-refresh-cmd</D:href>
            </D:activity-set> </D:prop>
            <D:status>HTTP/1.1 200 OK</D:status>
        </D:propstat> </D:response>
        </D:version-set> </D:prop>
        <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat> </D:response>
    </D:version-history> </D:prop>
    <D:status>HTTP/1.1 200 OK</D:status>
</D:propstat> </D:response>
</D:multistatus>

```

In this example, the DAV:creator-displayname and DAV:activity-set properties of the versions in the DAV:version-set of the DAV:version-history of <http://www.webdav.org/foo.html> are reported.

3.9 Additional OPTIONS Semantics

If the server supports the version-control feature, it MUST include "version-control" as a field in the DAV response header from an OPTIONS request on any resource that supports any versioning properties, reports, or methods.

3.10 Additional PUT Semantics

Additional Preconditions:

(DAV:cannot-modify-version-controlled-content): If the request-URL identifies a resource with a DAV:checked-in property, the request MUST fail unless DAV:auto-version semantics will automatically check out the resource.

(DAV:cannot-modify-version): If the request-URL identifies a version, the request MUST fail.

If the request creates a new resource that is automatically placed under version control, all preconditions for VERSION-CONTROL apply to the request.

Additional Postconditions:

(DAV:auto-checkout): If the resource was a checked-in version-controlled resource whose DAV:auto-version property indicates it should be automatically checked out but not automatically checked in for a modification request, then the server MUST have automatically checked out the resource prior to executing the request. In particular, the value of the DAV:checked-out property of the resource MUST be that of the DAV:checked-in property prior to the request, the DAV:checked-in property MUST have been removed, and the DAV:predecessor-set property MUST be initialized to be the same as the DAV:checked-out property. If any part of the checkout/update sequence failed, the status from the failed part of the request MUST be returned, and the server state preceding the request sequence MUST be restored.

(DAV:auto-checkout-checkin): If the resource was a checked-in version-controlled resource whose DAV:auto-version property indicates it should be automatically checked out and automatically checked in for a modification request, then the server MUST have automatically checked out the resource prior to executing the request and automatically checked it in after the request. In particular, the DAV:checked-in property of the resource MUST identify a new version whose content and dead properties are the same as those of the resource. The DAV:predecessor-set of the new version MUST identify the version identified by the DAV:checked-in property prior to the request. If any part of the checkout/update/checkin sequence failed, the status from the failed part of the request MUST be returned, and the server state preceding the request sequence MUST be restored.

If the request creates a new resource, the new resource MAY have automatically been placed under version control, and all postconditions for VERSION-CONTROL apply to the request.

3.11 Additional PROPFIND Semantics

A DAV:allprop PROPFIND request SHOULD NOT return any of the properties defined by this document. This allows a versioning server to perform efficiently when a naive client, which does not understand the cost of asking a server to compute all possible live properties, issues a DAV:allprop PROPFIND request.

Additional Preconditions:

(DAV:supported-live-property): If the request attempts to access a property defined by this document, the semantics of that property MUST be supported by the server.

3.12 Additional PROPPATCH Semantics

Additional Preconditions:

(DAV:cannot-modify-version-controlled-property): If the request attempts to modify a dead property, same semantics as PUT (see Section 3.10).

(DAV:cannot-modify-version): If the request attempts to modify a dead property, same semantics as PUT (see Section 3.10).

(DAV:cannot-modify-protected-property): An attempt to modify a property that is defined by this document, as being protected for that kind of resource, MUST fail.

(DAV:supported-live-property): An attempt to modify a property defined by this document, but whose semantics are not enforced by the server, MUST fail. This helps ensure that a client will be notified when it is trying to use a property whose semantics are not supported by the server.

Additional Postconditions:

(DAV:auto-checkout): If the request modified a dead property, same semantics as PUT (see Section 3.10).

(DAV:auto-checkout-checkin): If the request modified a dead property, same semantics as PUT (see Section 3.10).

3.13 Additional DELETE Semantics

Additional Preconditions:

(DAV:no-version-delete): A server MAY fail an attempt to DELETE a version.

Additional Postconditions:

(DAV:update-predecessor-set): If a version was deleted, the server MUST have replaced any reference to that version in a DAV:predecessor-set by a copy of the DAV:predecessor-set of the deleted version.

3.14 Additional COPY Semantics

Additional Preconditions:

If the request creates a new resource that is automatically placed under version control, all preconditions for VERSION-CONTROL apply to the request.

Additional Postconditions:

(DAV:must-not-copy-versioning-property): A property defined by this document MUST NOT have been copied to the new resource created by this request, but instead that property of the new resource MUST have the default initial value it would have had if the new resource had been created by a non-versioning method such as PUT or a MKCOL.

(DAV:auto-checkout): If the destination is a version-controlled resource, same semantics as PUT (see Section 3.10).

(DAV:auto-checkout-checkin): If the destination is a version-controlled resource, same semantics as PUT (see Section 3.10).

(DAV:copy-creates-new-resource): If the source of a COPY is a version-controlled resource or version, and if there is no resource at the destination of the COPY, then the COPY creates a new non-version-controlled resource at the destination of the COPY. The new resource MAY automatically be put under version control, but the resulting version-controlled resource MUST be associated with a new version history created for that new version-controlled resource, and all postconditions for VERSION-CONTROL apply to the request.

3.15 Additional MOVE Semantics

Additional Preconditions:

(DAV:cannot-rename-version): If the request-URL identifies a version, the request MUST fail.

Additional Postconditions:

(DAV:preserve-versioning-properties): When a resource is moved from a source URL to a destination URL, a property defined by this document MUST have the same value at the destination URL as it had at the source URL.

3.16 Additional UNLOCK Semantics

Note that these semantics apply both to an explicit UNLOCK request, as well as to the removal of a lock because of a lock timeout. If a precondition or postcondition cannot be satisfied, the lock timeout MUST NOT occur.

Additional Preconditions:

(DAV:version-history-is-tree): If the request-URL identifies a checked-out version-controlled resource that will be automatically checked in when the lock is removed, then the versions identified by the DAV:predecessor-set of the checked-out resource MUST be descendants of the root version of the version history for the DAV:checked-out version.

Additional Postconditions:

(DAV:auto-checkin): If the request-URL identified a checked-out version-controlled resource that had been automatically checked out because of its DAV:auto-version property, the request MUST have created a new version in the version history of the DAV:checked-out version. The request MUST have allocated a URL for the version that MUST NOT have previously identified any other resource, and MUST NOT ever identify a resource other than this version. The content, dead properties, DAV:resourcetype, and DAV:predecessor-set of the new version MUST be copied from the checked-out resource. The DAV:version-name of the new version MUST be set to a server-defined value distinct from all other DAV:version-name values of other versions in the same version history. The request MUST have removed the DAV:checked-out property of the version-controlled resource, and MUST have added a DAV:checked-in property that identifies the new version.

4 CHECKOUT-IN-PLACE FEATURE

With the version-control feature, WebDAV locking can be used to avoid the proliferation of versions that would result if every modification to a version-controlled resource produced a new version. The checkout-in-place feature provides an alternative mechanism that allows a client to explicitly check out and check in a resource to create a new version.

4.1 Additional Version Properties

The checkout-in-place feature introduces the following REQUIRED properties for a version.

4.1.1 DAV:checkout-fork

This property controls the behavior of CHECKOUT when a version already is checked out or has a successor. If the DAV:checkout-fork of a version is DAV:forbidden, a CHECKOUT request MUST fail if it would result in that version appearing in the DAV:predecessor-set or DAV:checked-out property of more than one version or checked-out resource. If DAV:checkout-fork is DAV:discouraged, such a CHECKOUT request MUST fail unless DAV:fork-ok is specified in the CHECKOUT request body.

A server MAY reject attempts to modify the DAV:checkout-fork of a version.

<!ELEMENT checkout-fork ANY>

ANY value: A sequence of elements with at most one DAV:discouraged or DAV:forbidden element.

<!ELEMENT discouraged EMPTY>

<!ELEMENT forbidden EMPTY>

4.1.2 DAV:checkin-fork

This property controls the behavior of CHECKIN when a version already has a successor. If the DAV:checkin-fork of a version is DAV:forbidden, a CHECKIN request MUST fail if it would result in that version appearing in the DAV:predecessor-set of more than one version. If DAV:checkin-fork is DAV:discouraged, such a CHECKIN request MUST fail unless DAV:fork-ok is specified in the CHECKIN request body.

A server MAY reject attempts to modify the DAV:checkout-fork of a version.

<!ELEMENT checkin-fork ANY>

ANY value: A sequence of elements with at most one DAV:discouraged or DAV:forbidden element.

<!ELEMENT discouraged EMPTY>

<!ELEMENT forbidden EMPTY>

4.2 Checked-Out Resource Properties

The checkout-in-place feature introduces the following REQUIRED properties for a checked-out resource.

4.2.1 DAV:checkout-fork

This property determines the DAV:checkout-fork property of the version that results from checking in this resource.

4.2.2 DAV:checkin-fork

This property determines the DAV:checkin-fork property of the version that results from checking in this resource.

4.3 CHECKOUT Method (applied to a version-controlled resource)

A CHECKOUT request can be applied to a checked-in version-controlled resource to allow modifications to the content and dead properties of that version-controlled resource.

If a CHECKOUT request fails, the server state preceding the request MUST be restored.

Marshalling:

If a request body is included, it MUST be a DAV:checkout XML element.

<!ELEMENT checkout ANY>

ANY value: A sequence of elements with at most one DAV:fork-ok element.

<!ELEMENT fork-ok EMPTY>

If a response body for a successful request is included, it MUST be a DAV:checkout-response XML element.

<!ELEMENT checkout-response ANY>

The response MUST include a Cache-Control:no-cache header.

Preconditions:

(DAV:must-be-checked-in): If a version-controlled resource is being checked out, it MUST have a DAV:checked-in property.

(DAV:checkout-of-version-with-descendant-is-forbidden): If the DAV:checkout-fork property of the version being checked out is DAV:forbidden, the request MUST fail if a version identifies that version in its DAV:predecessor-set.

(DAV:checkout-of-version-with-descendant-is-discouraged): If the DAV:checkout-fork property of the version being checked out is DAV:discouraged, the request MUST fail if a version identifies that version in its DAV:predecessor-set unless DAV:fork-ok is specified in the request body.

(DAV:checkout-of-checked-out-version-is-forbidden): If the DAV:checkout-fork property of the version being checked out is DAV:forbidden, the request MUST fail if a checked-out resource identifies that version in its DAV:checked-out property.

(DAV:checkout-of-checked-out-version-is-discouraged): If the DAV:checkout-fork property of the version being checked out is DAV:discouraged, the request MUST fail if a checked-out resource identifies that version in its DAV:checked-out property unless DAV:fork-ok is specified in the request body.

Postconditions:

(DAV:is-checked-out): The checked-out resource MUST have a DAV:checked-out property that identifies the DAV:checked-in version preceding the checkout. The version-controlled resource MUST NOT have a DAV:checked-in property.

(DAV:initialize-predecessor-set): The DAV:predecessor-set property of the checked-out resource MUST be initialized to be the DAV:checked-out version.

4.3.1 Example - CHECKOUT of a version-controlled resource

>>REQUEST

```
CHECKOUT /foo.html HTTP/1.1
Host: www.webdav.org
Content-Length: 0
```

>>RESPONSE

```
HTTP/1.1 200 OK
Cache-Control: no-cache
```

In this example, the version-controlled resource /foo.html is checked out.

4.4 CHECKIN Method (applied to a version-controlled resource)

A CHECKIN request can be applied to a checked-out version-controlled resource to produce a new version whose content and dead properties are copied from the checked-out resource.

If a CHECKIN request fails, the server state preceding the request MUST be restored.

Marshalling:

If a request body is included, it MUST be a DAV:checkin XML element.

<!ELEMENT checkin ANY>

ANY value: A sequence of elements with at most one DAV:keep-checked-out element and at most one DAV:fork-ok element.

<!ELEMENT keep-checked-out EMPTY>

<!ELEMENT fork-ok EMPTY>

If a response body for a successful request is included, it MUST be a DAV:checkin-response XML element.

<!ELEMENT checkin-response ANY>

The response MUST include a Cache-Control:no-cache header.

Preconditions:

(DAV:must-be-checked-out): The request-URL MUST identify a resource with a DAV:checked-out property.

(DAV:version-history-is-tree) The versions identified by the DAV:predecessor-set of the checked-out resource MUST be descendants of the root version of the version history for the DAV:checked-out version.

(DAV:checkin-fork-forbidden): A CHECKIN request MUST fail if it would cause a version whose DAV:checkin-fork is DAV:forbidden to appear in the DAV:predecessor-set of more than one version.

(DAV:checkin-fork-discouraged): A CHECKIN request MUST fail if it would cause a version whose DAV:checkin-fork is DAV:discouraged to appear in the DAV:predecessor-set of more than one version, unless DAV:fork-ok is specified in the request body.

Postconditions:

(DAV:create-version): The request MUST have created a new version in the version history of the DAV:checked-out version. The request MUST have allocated a distinct new URL for the new

version, and that URL MUST NOT ever identify any resource other than that version. The URL for the new version MUST be returned in a Location response header.

(DAV:initialize-version-content-and-properties): The content, dead properties, DAV:resourcetype, and DAV:predecessor-set of the new version MUST be copied from the checked-out resource. The DAV:version-name of the new version MUST be set to a server-defined value distinct from all other DAV:version-name values of other versions in the same version history.

(DAV:checked-in): If the request-URL identifies a version-controlled resource and DAV:keep-checked-out is not specified in the request body, the DAV:checked-out property of the version-controlled resource MUST have been removed and a DAV:checked-in property that identifies the new version MUST have been added.

(DAV:keep-checked-out): If DAV:keep-checked-out is specified in the request body, the DAV:checked-out property of the checked-out resource MUST have been updated to identify the new version.

4.4.1 Example - CHECKIN

>>REQUEST

```
CHECKIN /foo.html HTTP/1.1
Host: www.webdav.org
Content-Length: 0
```

>>RESPONSE

```
HTTP/1.1 201 Created
Location: http://repo.webdav.org/his/23/ver/32
Cache-Control: no-cache
```

In this example, version-controlled resource /foo.html is checked in, and a new version is created at <http://repo.webdav.org/his/23/ver/32>.

4.5 UNCHECKOUT Method

An UNCHECKOUT request can be applied to a checked-out version-controlled resource to cancel the CHECKOUT and restore the pre-CHECKOUT state of the version-controlled resource.

If an UNCHECKOUT request fails, the server MUST undo any partial effects of the UNCHECKOUT request.

Marshalling:

If a request body is included, it MUST be a DAV:uncheckout XML element.

```
<!ELEMENT uncheckout ANY>
```

If a response body for a successful request is included, it MUST be a DAV:uncheckout-response XML element.

```
<!ELEMENT uncheckout-response ANY>
```

The response MUST include a Cache-Control:no-cache header.

Preconditions:

(DAV:must-be-checked-out-version-controlled-resource): The request-URL MUST identify a version-controlled resource with a DAV:checked-out property.

Postconditions:

(DAV:cancel-checked-out): The value of the DAV:checked-in property is that of the DAV:checked-out property prior to the request, and the DAV:checked-out property has been removed.

(DAV:restore-content-and-dead-properties): The content and dead properties of the version-controlled resource are copies of its DAV:checked-in version.

4.5.1 Example - UNCHECKOUT

>>REQUEST

```
UNCHECKOUT /foo.html HTTP/1.1
Host: www.webdav.org
Content-Length: 0
```

>>RESPONSE

```
HTTP/1.1 200 OK
Cache-Control: no-cache
```

In this example, the content and dead properties of the version-controlled resource identified by `http://www.webdav.org/foo.html` are restored to their values preceding the most recent CHECKOUT of that version-controlled resource.

4.6 Additional OPTIONS Semantics

If a server supports the checkout-in-place feature, it MUST include "checkout-in-place" as a field in the DAV response header from an OPTIONS request on any resource that supports any versioning properties, reports, or methods.

5 Version-History Feature

It is often useful to have access to a version history even after all version-controlled resources for that version history have been deleted. A server can provide this functionality by supporting version history resources. A version history resource is a resource that exists in a server defined namespace and therefore is unaffected by any deletion or movement of version-controlled resources. A version history resource is an appropriate place to add a property that logically applies to all states of a resource. The DAV:expand-property report (see Section 3.8) can be applied to the DAV:version-set of a version history resource to provide a variety of useful reports on all versions in that version history.

5.1 Version History Properties

The DAV:resourcetype of a version history MUST be DAV:version-history.

The version-history feature introduces the following REQUIRED properties for a version history.

5.1.1 DAV:version-set (protected)

This property identifies each version of this version history.

```
<!ELEMENT version-set (href+)>
```

5.1.2 DAV:root-version (computed)

This property identifies the root version of this version history.

```
<!ELEMENT root-version (href)>
```

5.2 Additional Version-Controlled Resource Properties

The version-history feature introduces the following REQUIRED property for a version-controlled resource.

5.2.1 DAV:version-history (computed)

This property identifies the version history resource for the DAV:checked-in or DAV:checked-out version of this version-controlled resource.

```
<!ELEMENT version-history (href)>
```

5.3 Additional Version Properties

The version-history feature introduces the following REQUIRED property for a version.

5.3.1 DAV:version-history (computed)

This property identifies the version history that contains this version.

```
<!ELEMENT version-history (href)>
```

5.4 DAV:locate-by-history Report

Many properties identify a version from some version history. It is often useful to be able to efficiently locate a version-controlled resource for that version history. The DAV:locate-by-history report can be applied to a collection to locate the collection member that is a version-controlled resource for a specified version history resource.

Marshalling:

The request body MUST be a DAV:locate-by-history XML element.

```
<!ELEMENT locate-by-history (version-history-set, prop)>
<!ELEMENT version-history-set (href+)>
prop: see RFC 2518, Section 12.11
```

The response body for a successful request MUST be a DAV:multistatus XML element containing every version-controlled resource that is a member of the collection identified by the request-URL, and whose DAV:version-history property identifies one of the version history resources identified by the request body. The DAV:prop element in the request body identifies which properties should be reported in the DAV:prop elements in the response body.

Preconditions:

(DAV:must-be-version-history): Each member of the DAV:version-history-set element in the request body MUST identify a version history resource.

5.4.1 Example - DAV:locate-by-history Report

>>REQUEST

```
REPORT /ws/public HTTP/1.1
Host: www.webdav.org
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:locate-by-history xmlns:D="DAV:">
  <D:version-history-set>
    <D:href>http://repo.webdav.org/his/23</D:href>
    <D:href>http://repo.webdav.org/his/84</D:href>
    <D:href>http://repo.webdav.org/his/129</D:href>
  </D:version-history-set/>
  <D:prop>
    </D:version-history>
  </D:prop>
</D:locate-by-history>
```

>>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>http://www.webdav.org/ws/public/x/test.html</D:href>
    <D:propstat>
      <D:prop>
        <D:version-history>
          <D:href>http://repo.webdav.org/his/23</D:href>
        </D:version-history>
      </D:prop>
      <D:status>HTTP/1.1 200 OK</D:status>
    </D:propstat>
  </D:response>
</D:multistatus>
```

In this example, there is only one version-controlled member of /ws/public that is a version-controlled resource for one of the three specified version history resources. In particular, /ws/public/x/test.html is the version-controlled resource for <http://repo.webdav.org/his/23>.

5.5 Additional OPTIONS Semantics

If the server supports the version-history feature, it MUST include "version-history" as a field in the DAV response header from an OPTIONS request on any resource that supports any versioning properties, reports, or methods.

A DAV:version-history-collection-set element MAY be included in the request body to identify collections that may contain version history resources.

Additional Marshalling:

If an XML request body is included, it MUST be a DAV:options XML element.

<!ELEMENT options ANY>

ANY value: A sequence of elements with at most one DAV:version-history-collection-set element.

If an XML response body for a successful request is included, it MUST be a DAV:options-response XML element.

<!ELEMENT options-response ANY>

ANY value: A sequence of elements with at most one DAV:version-history-collection-set element.

<!ELEMENT version-history-collection-set (href*)>

If DAV:version-history-collection-set is included in the request body, the response body for a successful request MUST contain a DAV:version-history-collection-set element identifying collections that may contain version histories. An identified collection MAY be the root collection of a tree of collections, all of which may contain version histories. Since different servers can control different parts of the URL namespace, different resources on the same host MAY have different DAV:version-history-collection-set values. The identified collections MAY be located on different hosts from the resource.

5.6 Additional DELETE Semantics

Additional Postconditions:

(DAV:delete-version-set): If the request deleted a version history, the request MUST have deleted all versions in the DAV:version-set of that version history, and MUST have satisfied the postconditions for version deletion (see Section 3.13).

(DAV:version-history-has-root): If the request deleted the root version of a version history, the request MUST have updated the DAV:root-version of the version history to refer to another version that is an ancestor of all other remaining versions in that version history. A result of this postcondition is that every version history will have at least one version, and the only way to delete all versions is to delete the version history resource.

5.7 Additional COPY Semantics

Additional Preconditions:

(DAV:cannot-copy-history): If the request-URL identifies a version history, the request MUST fail. In order to create another version history whose versions have the same content and dead properties, the appropriate sequence of VERSION-CONTROL, CHECKOUT, PUT, PROPPATCH, and CHECKIN requests must be made.

5.8 Additional MOVE Semantics

Additional Preconditions:

(DAV:cannot-rename-history): If the request-URL identifies a version history, the request MUST fail.

5.9 Additional VERSION-CONTROL Semantics

Additional Postconditions:

(DAV:new-version-history): If the request created a new version history, the request MUST have allocated a new server-defined URL for that version history that MUST NOT have previously identified any other resource, and MUST NOT ever identify a resource other than this version history.

5.10 Additional CHECKIN Semantics

Additional Postconditions:

(DAV:add-to-history): A URL for the new version resource MUST have been added to the DAV:version-set of the version history of the DAV:checked-out version.

6 Workspace Feature

In order to allow multiple users to work concurrently on adding versions to the same version history, it is necessary to allocate on the server multiple checked-out resources for the same version history. Even if only one user is making changes to a resource, that user will sometimes wish to create a "private" version, and then to expose that version at a later time. One way to provide this functionality depends on the client keeping track of its current set of checked-out resources. This is the working-resource feature defined in Section 8. The other way to provide this functionality avoids the need for persistent state on the client, and instead has the server maintain a human meaningful namespace for related sets of checked-out resources. This is the workspace feature defined in this section.

The workspace feature introduces a "workspace resource". A workspace resource is a collection whose members are related version-controlled and non-version-controlled resources. Multiple workspaces may be used to expose different versions and configurations of a set of version-controlled resources concurrently. In order to make changes to a version-controlled resource in one workspace visible in another workspace, that version-controlled resource must be checked in, and then the corresponding version-controlled resource in the other workspace can be updated to display the content and dead properties of the new version.

In order to ensure unambiguous merging (see Section 11) and baselining (see Section 12) semantics, a workspace may contain at most one version-controlled resource for a given version history. This is required for unambiguous merging because the MERGE method must identify which version-controlled resource is to be the merge target of a given version. This is required for unambiguous baselining because a baseline can only select one version for a given version-controlled resource.

Initially, an empty workspace can be created. Non-version-controlled resources can then be added to the workspace with standard WebDAV requests such as PUT and MKCOL. Version-controlled resources can be added to the workspace with VERSION-CONTROL requests. If the

baseline feature is supported, collections in the workspace can be placed under baseline control, and then initialized by existing baselines.

6.1 Workspace Properties

The workspace feature introduces the following REQUIRED property for a workspace.

6.1.1 DAV:workspace-checkout-set (computed)

This property identifies each checked-out resource whose DAV:workspace property identifies this workspace.

```
<!ELEMENT workspace-checkout-set (href*)>
```

6.2 Additional Resource Properties

The workspace feature introduces the following REQUIRED property for a WebDAV resource.

6.2.1 DAV:workspace (protected)

The DAV:workspace property of a workspace resource MUST identify itself. The DAV:workspace property of any other type of resource MUST be the same as the DAV:workspace of its parent collection.

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

6.3 MKWORKSPACE Method

A MKWORKSPACE request creates a new workspace resource. A server MAY restrict workspace creation to particular collections, but a client can determine the location of these collections from a DAV:workspace-collection-set OPTIONS request (see Section 6.4).

If a MKWORKSPACE request fails, the server state preceding the request MUST be restored.

Marshalling:

If a request body is included, it MUST be a DAV:mkworkspace XML element.

```
<!ELEMENT mkworkspace ANY>
```

If a response body for a successful request is included, it MUST be a DAV:mkworkspace-response XML element.

<!ELEMENT mkworkspace-response ANY>

The response MUST include a Cache-Control:no-cache header.

Preconditions:

(DAV:resource-must-be-null): A resource MUST NOT exist at the request-URL.

(DAV:workspace-location-ok): The request-URL MUST identify a location where a workspace can be created.

Postconditions:

(DAV:initialize-workspace): A new workspace exists at the request-URL. The DAV:resourcetype of the workspace MUST be DAV:collection. The DAV:workspace of the workspace MUST identify the workspace.

6.3.1 Example - MKWORKSPACE

>>REQUEST

MKWORKSPACE /ws/public HTTP/1.1
Host: www.webdav.org
Content-Length: 0

>>RESPONSE

HTTP/1.1 201 Created
Cache-Control: no-cache

In this example, a new workspace is created at
<http://www.webdav.org/ws/public>.

6.4 Additional OPTIONS Semantics

If a server supports the workspace feature, it MUST include "workspace" as a field in the DAV response header from an OPTIONS request on any resource that supports any versioning properties, reports, or methods.

If a server supports the workspace feature, it MUST also support the checkout-in-place feature and the version-history feature.

A DAV:workspace-collection-set element MAY be included in the request body to identify collections that may contain workspace resources.

Additional Marshalling:

If an XML request body is included, it MUST be a DAV:options XML element.

<!ELEMENT options ANY>

ANY value: A sequence of elements with at most one DAV:workspace-collection-set element.

If an XML response body for a successful request is included, it MUST be a DAV:options-response XML element.

<!ELEMENT options-response ANY>

ANY value: A sequence of elements with at most one DAV:workspace-collection-set element.

<!ELEMENT workspace-collection-set (href*)>

If DAV:workspace-collection-set is included in the request body, the response body for a successful request MUST contain a DAV:workspace-collection-set element identifying collections that may contain workspaces. An identified collection MAY be the root collection of a tree of collections, all of which may contain workspaces. Since different servers can control different parts of the URL namespace, different resources on the same host MAY have different DAV:workspace-collection-set values. The identified collections MAY be located on different hosts from the resource.

6.4.1 Example - OPTIONS

>>REQUEST

```
OPTIONS /doc HTTP/1.1
Host: www.webdav.org
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx
```

```
<?xml version="1.0" encoding="utf-8" ?>
<D:options xmlns:D="DAV:">
  <D:version-history-collection-set/>
  <D:workspace-collection-set/>
</D:options>
```

>>RESPONSE

```
HTTP/1.1 200 OK
DAV: 1
DAV: version-control,checkout-in-place,version-history,workspace
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx
```

```
<?xml version="1.0" encoding="utf-8" ?>
<D:options-response xmlns:D="DAV:">
  <D:version-history-collection-set>
    <D:href>http://repo.webdav.org/his</D:href>
  </D:version-history-collection-set>
  <D:workspace-collection-set>
    <D:href>http://www.webdav.org/public/ws</D:href>
    <D:href>http://www.webdav.org/private/ws</D:href>
  </D:workspace-collection-set>
</D:options-response>
```

In this example, the server indicates that it provides Class 1 DAV support and basic-server-workspace versioning support. In addition, the server indicates the requested locations of the version history resources and the workspace resources.

6.5 Additional DELETE Semantics

Additional Postconditions:

(DAV:delete-workspace-members): If a workspace is deleted, any resource that identifies that workspace in its DAV:workspace property MUST be deleted.

6.6 Additional MOVE Semantics

Additional Postconditions:

(DAV:workspace-member-moved): If the request-URL did not identify a workspace, the DAV:workspace of the destination MUST have been updated to have the same value as the DAV:workspace of the parent collection of the destination.

(DAV:workspace-moved): If the request-URL identified a workspace, any reference to that workspace in a DAV:workspace property MUST have been updated to refer to the new location of that workspace.

6.7 Additional VERSION-CONTROL Semantics

A VERSION-CONTROL request can be used to create a new version-controlled resource for an existing version history. This allows the creation of version-controlled resources for the same version history in multiple workspaces.

Additional Marshalling:

<!ELEMENT version-control ANY>

ANY value: A sequence of elements with at most one DAV:version element.

<!ELEMENT version (href)>

Additional Preconditions:

(DAV:cannot-add-to-existing-history): If the DAV:version-control request body element contains a DAV:version element, the request-URL MUST NOT identify a resource.

(DAV:must-be-version): The DAV:href of the DAV:version element MUST identify a version.

(DAV:one-version-controlled-resource-per-history-per-workspace): If the DAV:version-control request body specifies a version, and if the request-URL is a member of a workspace, then there MUST NOT already be a version-controlled member of that workspace whose DAV:checked-in or DAV:checked-out property identifies any version from the version history of the version specified in the request body.

Additional Postconditions:

(DAV:new-version-controlled-resource): If the request-URL did NOT identify a resource, a new version-controlled resource exists at the request-URL whose content and dead properties are initialized by those of the version in the request body, and whose DAV:checked-in property identifies that version.

6.7.1 Example - VERSION-CONTROL (using an existing version history)

>>REQUEST

```
VERSION-CONTROL /ws/public/bar.html HTTP/1.1
Host: www.webdav.org
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx
```

```
<?xml version="1.0" encoding="utf-8" ?>
<D:version-control xmlns:D="DAV:">
  <D:version>
    <D:href>http://repo.webdav.org/his/12/ver/V3</D:href>
  </D:version>
</D:version-control>
```

>>RESPONSE

```
HTTP/1.1 201 Created
Cache-Control: no-cache
```

In this example, a new version-controlled resource is created at /ws/public/bar.html. The content and dead properties of the new version-controlled resource are initialized to be the same as those of the version identified by <http://repo.webdav.org/his/12/ver/V3>.

7 UPDATE Feature

The update feature provides a mechanism for changing the state of a checked-in version-controlled resource to be that of another version from the version history of that resource.

7.1 UPDATE Method

The UPDATE method modifies the content and dead properties of a checked-in version-controlled resource (the "update target") to be those of a specified version (the "update source") from the version history of that version-controlled resource.

The response to an UPDATE request identifies the resources modified by the request, so that a client can efficiently update any cached state it is maintaining. Extensions to the UPDATE method allow multiple resources to be modified from a single UPDATE request (see Section 12.13).

Marshalling:

The request body MUST be a DAV:update element.

<!ELEMENT update ANY>

ANY value: A sequence of elements with at most one DAV:version element and at most one DAV:prop element.

<!ELEMENT version (href)>

prop: see RFC 2518, Section 12.11

The response for a successful request MUST be a 207 Multi-Status, where the DAV:multistatus XML element in the response body identifies all resources that have been modified by the request.

multistatus: see RFC 2518, Section 12.9

The response MUST include a Cache-Control:no-cache header.

Postconditions:

(DAV:update-content-and-properties): If the DAV:version element in the request body identified a version that is in the same version history as the DAV:checked-in version of a version-controlled resource identified by the request-URL, then the content and dead properties of that version-controlled resource MUST be the same as those of the version specified by the DAV:version element, and the DAV:checked-in property of the version-controlled resource MUST identify that version. The request-URL MUST appear in a DAV:response element in the response body.

(DAV:report-properties): If DAV:prop is specified in the request body, the properties specified in the DAV:prop element MUST be reported in the DAV:response elements in the response body.

7.1.1 Example - UPDATE

>>REQUEST

```
UPDATE /foo.html HTTP/1.1
Host: www.webdav.org
Content-type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:update xmlns:D="DAV:">
  <D:version>
    <D:href>http://repo.webdav.org/his/23/ver/33</D:href>
  </D:version>
</D:update>
```

>>RESPONSE

```
HTTP/1.1 207 Multi-Status
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx
Cache-Control: no-cache

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
    <D:href>http://www.webdav.org/foo.html</D:href>
    <D:status>HTTP/1.1 200 OK</D:status>
  </D:response>
```

In this example, the content and dead properties of `http://repo.webdav.org/his/23/ver/33` are copied to the version-controlled resource `/foo.html`, and the `DAV:checked-in` property of `/foo.html` is updated to refer to `http://repo.webdav.org/his/23/ver/33`.

7.2 Additional OPTIONS Semantics

If the server supports the update feature, it MUST include "update" as a field in the DAV response header from an OPTIONS request on any resource that supports any versioning properties, reports, or methods.

8 Label Feature

A version "label" is a string that distinguishes one version in a version history from all other versions in that version history. A label can automatically be assigned by a server, or it can be assigned by a client in order to provide a meaningful name for that version. A given version label can be assigned to at most one version of a given version history, but client assigned labels can be reassigned to another version at any time. Note that although a given label can be applied to at most one version from the same version history, the same label can be applied to versions from different version histories.

For certain methods, if the request-URL identifies a version-controlled resource, a label can be specified in a Label request header (see Section 8.3) to cause the method to be applied to the version selected by that label from the version history of that version-controlled resource.

8.1 Additional Version Properties

The label feature introduces the following REQUIRED property for a version.

8.1.1 DAV:label-name-set (protected)

This property contains the labels that currently select this version.

```
<!ELEMENT label-name-set (label-name*)>
<!ELEMENT label-name (#PCDATA)>
PCDATA value: string
```

8.2 LABEL Method

A LABEL request can be applied to a version to modify the labels that select that version. The case of a label name MUST be preserved when it is stored and retrieved. When comparing two label names to decide if they match or not, a server SHOULD use a case-sensitive URL-escaped UTF-8 encoded comparison of the two label names.

If a LABEL request is applied to a checked in version-controlled resource, the operation MUST be applied to the DAV:checked-in version of that version-controlled resource.

Marshalling:

The request body **MUST** be a DAV:label element.

<!ELEMENT label ANY>

ANY value: A sequence of elements with at most one DAV:add, DAV:set, or DAV:remove element.

<!ELEMENT add (label-name)>

<!ELEMENT set (label-name)>

<!ELEMENT remove (label-name)>

<!ELEMENT label-name (#PCDATA)>

PCDATA value: string

The request **MAY** include a Label header.

The request **MAY** include a Depth header. If no Depth header is included, Depth:0 is assumed. Standard depth semantics apply, and the request is applied to the collection identified by the request-URL and to all members of the collection that satisfy the Depth value. If a Depth header is included and the request fails on any resource, the response **MUST** be a 207 Multi-Status that identifies all resources for which the request has failed.

If a response body for a successful request is included, it **MUST** be a DAV:label-response XML element.

<!ELEMENT label-response ANY>

The response **MUST** include a Cache-Control:no-cache header.

Preconditions:

(DAV:must-be-checked-in): If the request-URL identifies a version-controlled resource, the version-controlled resource **MUST** be checked in.

(DAV:must-select-version-in-history): If a Label request header is included and the request-URL identifies a version-controlled resource, the specified label **MUST** select a version in the version history of the version-controlled resource.

(DAV:add-must-be-new-label): If DAV:add is specified in the request body, the specified label **MUST NOT** appear in the DAV:label-name-set of any version in the version history of that version-controlled resource.

(DAV:label-must-exist): If DAV:remove is specified in the request body, the specified label MUST appear in the DAV:label-name-set of that version.

Postconditions:

(DAV:add-or-set-label): If DAV:add or DAV:set is specified in the request body, the specified label MUST appear in the DAV:label-name-set of the specified version, and MUST NOT appear in the DAV:label-name-set of any other version in the version history of that version.

(DAV:remove-label): If DAV:remove is specified in the request body, the specified label MUST NOT appear in the DAV:label-name-set of any version in the version history of that version.

8.2.1 Example - Setting a label

>>REQUEST

```

LABEL /foo.html HTTP/1.1
Host: www.webdav.org
Content-type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:label xmlns:D="DAV:">
  <D:set>
    <D:label-name>default</D:label-name>
  </D:set>
</D:label>
```

>>RESPONSE

```

HTTP/1.1 200 OK
Cache-Control: no-cache
```

In this example, the label "default" is applied to the DAV:checked-in version of /foo.html.

8.3 Label Header

For certain methods (e.g. GET, PROPFIND), if the request-URL identifies a version-controlled resource, a label can be specified in a Label request header to cause the method to be applied to the version selected by that label from the version history of that version-controlled resource.

The value of a label header is the name of a label, encoded using URL-escaped UTF-8. For example, the label "release B.3" is identified by the following header:

```
Label: release%20B.3
```

A Label header MUST have no effect on a request whose request-URL does not identify a version-controlled resource. In particular, it MUST have no effect on a request whose request-URL identifies a version or a version history.

A server MUST return an HTTP-1.1 Vary header containing Label in a successful response to a cacheable request (e.g., GET) that includes a Label header.

8.4 Additional OPTIONS Semantics

If the server supports the label feature, it MUST include "label" as a field in the DAV response header from an OPTIONS request on any resource that supports any versioning properties, reports, or methods.

8.5 Additional GET Semantics

Additional Marshalling:

The request MAY include a Label header.

Additional Preconditions:

(DAV:must-select-version-in-history): If a Label request header is included and the request-URL identifies a version-controlled resource, the specified label MUST select a version in the version history of the version-controlled resource.

Additional Postconditions:

(DAV:apply-request-to-labeled-version): If the request-URL identifies a version-controlled resource and a Label request header is included, the response MUST contain the content of the specified version rather than that of the version-controlled resource.

8.6 Additional PROPFIND Semantics

Additional Marshalling:

The request MAY include a Label header.

Additional Preconditions:

(DAV:must-select-version-in-history): If a Label request header is included and the request-URL identifies a version-controlled resource, the specified label MUST select a version in the version history of the version-controlled resource.

Additional Postconditions:

(DAV:apply-request-to-labeled-version): If the request-URL identifies a version-controlled resource and a Label request header is included, the response MUST contain the properties of the specified version rather than that of the version-controlled resource.

8.7 Additional COPY Semantics

Additional Marshalling:

The request MAY include a Label header.

Additional Preconditions:

(DAV:must-select-version-in-history): If a Label request header is included and the request-URL identifies a version-controlled resource, the specified label MUST select a version in the version history of the version-controlled resource.

Additional Postconditions:

(DAV:apply-request-to-labeled-version): If the request-URL identifies a version-controlled resource and a Label request header is included, the request MUST have copied the properties and content of the specified version rather than that of the version-controlled resource.

8.8 Additional CHECKOUT Semantics

If the server supports the working-resource option, a LABEL header may be included to check out the version selected by the specified label.

Additional Marshalling:

The request MAY include a Label header.

Additional Preconditions:

(DAV:must-select-version-in-history): If a Label request header is included and the request-URL identifies a version-controlled resource, the specified label MUST select a version in the version history of the version-controlled resource.

(DAV:must-not-have-label-and-apply-to-version): If a Label request header is included, the request body MUST NOT contain a DAV:apply-to-version element.

Additional Postconditions:

(DAV:apply-request-to-labeled-version): If the request-URL identifies a checked-in version-controlled resource, and a Label request header is included, the CHECKOUT MUST have been applied to the version selected by the specified label, and not to the version-controlled resource itself.

8.9 Additional UPDATE Semantics

If the request body of an UPDATE request contains a DAV:label-name element, the update target is the resource identified by the request-URL, and the update source is the version selected by the specified label from the version history of the update target.

Additional Marshalling:

```
<!ELEMENT update ANY>
ANY value: A sequence of elements with at most one DAV:label-name
or DAV:version element (but not both).
<!ELEMENT label-name (#PCDATA)>
PCDATA value: string
```

The request MAY include a Depth header. If no Depth header is included, Depth:0 is assumed. Standard depth semantics apply, and the request is applied to the collection identified by the request-URL and to all members of the collection that satisfy the Depth value. If a Depth header is included and the request fails on any resource, the response MUST be a 207 Multi-Status that identifies all resources for which the request has failed.

Additional Preconditions:

(DAV:must-select-version-in-history): If the request includes a DAV:label-name element in the request body, the label MUST select a version in the version history of the version-controlled resource identified by the request-URL.

(DAV:depth-update): If the request includes a Depth header, standard depth semantics apply, and the request is applied to the collection identified by the request-URL and to all members of the collection that satisfy the Depth value. The request **MUST** be applied to a collection before being applied to any members of that collection, since an update of a version-controlled collection might change the membership of that collection.

Additional Postconditions:

(DAV:apply-request-to-labeled-version): If a DAV:label-name element appears in the request body, the content and dead properties of the version-controlled resource must have been updated to be those of the version selected by that label.

9 Working-Resource Feature

The working-resource feature provides an alternative to the workspace feature for supporting parallel development. Unlike the workspace feature, where the desired configuration of versions and checked-out resources is maintained on the server, the working-resource feature maintains the configuration on the client. This simplifies the server implementation, but does not allow a user to access the configuration from clients in different physical locations, such as from another office, from home, or while traveling. Another difference is that the workspace feature isolates clients from a logical change that involves renaming shared resources, until that logical change is complete and tested; with the working resource feature, all clients use a common set of shared version-controlled resources and every client sees the result of a MOVE as soon as it occurs.

If a server supports the working-resource feature but not the checkout-in-place feature, a CHECKOUT request can only be used to create a working resource, and cannot be used to check out a version-controlled resource. If a server supports the checkout-in-place feature, but not the working-resource feature, a CHECKOUT can only be used to change the state of a version-controlled resource from checked-in to checked-out.

9.1 Additional Version Properties

The working-resource feature introduces the following **REQUIRED** properties for a version.

9.1.1 DAV:checkout-fork

This property is defined in Section 4.1.1.

9.1.2 DAV:checkin-fork

This property is defined in Section 4.1.2.

9.2 Working Resource Properties

The working-resource feature introduces the following REQUIRED properties for a working resource. Since a working resource is a checked-out resource, it also has any property defined in this document for a checked-out resource.

9.2.1 DAV:auto-update (protected)

This property identifies the version-controlled resource that will be updated when the working resource is checked in.

```
<!ELEMENT auto-update (href)>
```

9.2.2 DAV:checkout-fork

This property is defined in Section 4.2.1.

9.2.3 DAV:checkin-fork

This property is defined in Section 4.2.2.

9.3 CHECKOUT Method (applied to a version)

A CHECKOUT request can be applied to a version to create a new working resource. The content and dead properties of the working resource are a copy of the version that was checked out.

Marshalling:

If a request body is included, it MUST be a DAV:checkout XML element.

```
<!ELEMENT checkout ANY>
```

ANY value: A sequence of elements with at most one DAV:apply-to-version and at most one DAV:fork-ok element.

```
<!ELEMENT apply-to-version EMPTY>
```

```
<!ELEMENT fork-ok EMPTY>
```

If a response body for a successful request is included, it MUST be a DAV:checkout-response XML element.

<!ELEMENT checkout-response ANY>

The response MUST include a Location header.

The response MUST include a Cache-Control:no-cache header.

Preconditions:

(DAV:checkout-of-version-with-descendant-is-forbidden): See Section 4.3.

(DAV:checkout-of-version-with-descendant-is-discouraged): See Section 4.3.

(DAV:checkout-of-checked-out-version-is-forbidden): See Section 4.3.

(DAV:checkout-of-checked-out-version-is-discouraged): See Section 4.3.

Postconditions:

(DAV:create-working-resource): If the request-URL identified a version, the Location response header MUST contain the URL of a new working resource. The DAV:checked-out property of the new working resource MUST identify the version that was checked out. The content and dead properties of the working resource MUST be copies of the content and dead properties of the DAV:checked-out version. The DAV:predecessor-set property of the working resource MUST be initialized to be the version identified by the request-URL. The DAV:auto-update property of the working resource MUST NOT exist.

(DAV:create-working-resource-from-checked-in-version): If the request-URL identified a version-controlled resource, and DAV:apply-to-version is specified in the request body, the CHECKOUT is applied to the DAV:checked-in version of the version-controlled resource, and not the version-controlled resource itself. A new working resource is created and the version-controlled resource remains checked-in. The DAV:auto-update property of the working resource MUST identify the version-controlled resource.

9.3.1 Example - CHECKOUT of a version

>>REQUEST

```
CHECKOUT /his/12/ver/V3 HTTP/1.1
Host: repo.webdav.org
Content-Length: 0
```

>>RESPONSE

```
HTTP/1.1 201 Created
Location: http://repo.webdav.org/wr/157
Cache-Control: no-cache
```

In this example, the version identified by `http://repo.webdav.org/his/12/ver/V3` is checked out, and the new working resource is located at `http://repo.webdav.org/wr/157`.

9.4 CHECKIN Method (applied to a working resource)

A CHECKIN request can be applied to a working resource to produce a new version whose content and dead properties are a copy of those of the working resource. If the DAV:auto-update property of the working resource was set because the working resource was created by applying a CHECKOUT with the DAV:apply-to-version flag to a version-controlled resource, the CHECKIN request will also update the content and dead properties of that version-controlled resource to be those of the new version.

Marshalling:

If a request body is included, it MUST be a DAV:checkin XML element.

<!ELEMENT checkin ANY>

ANY value: A sequence of elements with at most one DAV:fork-ok element.

<!ELEMENT fork-ok EMPTY>

If a response body for a successful request is included, it MUST be a DAV:checkin-response XML element.

<!ELEMENT checkin-response ANY>

The response MUST include a Cache-Control:no-cache header.

Preconditions:

(DAV:must-be-checked-out): See Section 4.4.

(DAV:version-history-is-tree) See Section 4.4.

(DAV:checkin-fork-forbidden): See Section 4.4.

(DAV:checkin-fork-discouraged): See Section 4.4.

(DAV:no-overwrite-by-auto-update): If the DAV:auto-update property for the checked-out resource identifies a version-controlled resource, at least one of the versions identified by the DAV:predecessor-set property of the checked-out resource MUST identify a version that is either the same as or a descendant of the version identified by the DAV:checked-in property of that version-controlled resource.

Postconditions:

(DAV:create-version): See Section 4.4.

(DAV:initialize-version-content-and-properties): See Section 4.4.

(DAV:auto-update): If the DAV:auto-update property of the checked-out resource identified a version-controlled resource, an UPDATE request with the new version MUST have been applied to that version-controlled resource.

(DAV:delete-working-resource): If the request-URL identifies a working resource and if DAV:keep-checked-out is not specified in the request body, the working resource is deleted.

9.4.1 Example - CHECKIN of a working resource

>>REQUEST

```
CHECKIN /wr/157 HTTP/1.1
Host: repo.webdav.org
Content-Length: 0
```

>>RESPONSE

```
HTTP/1.1 201 Created
Location: http://repo.webdav.org/his/23/ver/15
Cache-Control: no-cache
```

In this example, the working resource /wr/157 checked in, and a new version is created at <http://repo.webdav.org/his/23/ver/15>.

9.5 Additional OPTIONS Semantics

If the server supports the working-resource feature, it MUST include "working-resource" as a field in the DAV response header from an OPTIONS request on any resource that supports any versioning properties, reports, or methods.

9.6 Additional COPY Semantics

Additional Postconditions:

(DAV:copy-creates-new-resource): The result of copying a working resource is a new non-version-controlled resource at the destination of the COPY. The new resource MAY automatically be put under version control, but the resulting version-controlled resource MUST be associated with a new version history created for that new version-controlled resource.

9.7 Additional MOVE Semantics

Additional Preconditions:

(DAV:cannot-rename-working-resource): If the request-URL identifies a working resource, the request MUST fail.

Additional Postconditions:

(DAV:update-auto-update): If the request-URL identified a version-controlled resource, any DAV:auto-update properties that identified that version-controlled resource MUST have been updated to contain the new location of that version-controlled resource.

10 Advanced Versioning Features

Advanced versioning addresses the problems of parallel development and configuration management of multiple sets of interrelated resources. Traditionally, artifacts of software development, including requirements, design documents, code, and test cases, have been a focus of configuration management. Web sites, comprising multiple inter-linked resources (HTML, graphics, sound, CGI, and others), are another class of complex information artifacts that benefit from the application of configuration management. The advanced versioning capabilities for coordinating concurrent change provide the infrastructure for efficient and controlled management of large evolving web sites.

10.1 Advanced Versioning Packages

Although a server MAY support any combination of advanced versioning features, in order to minimize the complexity of a WebDAV advanced versioning client, a WebDAV advanced versioning server SHOULD support one of the following packages:

Advanced-Server-Workspace Package: basic-server-workspace package plus all advanced features

Advanced-Client-Workspace Package: basic-client-workspace package plus all advanced features

The advanced-server-workspace package supports advanced versioning capabilities for a client with no persistent state. The advanced-client-workspace package supports advanced versioning capabilities for a client that maintains configuration state on the client. A server that supports both advanced workspace packages will interoperate with all versioning clients.

10.2 Advanced Versioning Terms

The following additional terms are used by the advanced versioning features.

Collection

A "collection" is a resource whose state consists of not only content and properties, but also a set of named "bindings", where a binding identifies what RFC 2518 calls an "internal member" of the collection. Note that a binding is not a resource, but rather is a part of the state of a collection that defines a mapping from a binding name (a URL segment) to a resource (an internal member of the collection).

Collection Version Resource

A "collection version resource", or simply "collection version", captures the dead properties of a version-controlled collection, as well as the names of its version-controlled bindings (see Section 14). A version-controlled binding is a binding to a version-controlled resource. If the checkout-in-place feature is supported, a collection version can be created by checking out and then checking in a version-controlled collection. If the working-resource feature is supported, a collection version can be created by checking out a collection version (to create a "working collection") and then checking in the working collection.

Configuration

A "configuration" is a set of resources that consists of a root collection and all members (not just internal members) of that root collection that are not members of another configuration. The root collection is called the "configuration root", and the members of this set are called the "members of the configuration". Note that a collection (which is a single resource) is very different from a configuration (which is a set of resources).

Baseline Resource

A "baseline resource", or simply "baseline", of a collection is a version of the configuration that is rooted at that collection (see Section 12). In particular, a baseline captures the DAV:checked-in version of every version-controlled member of that configuration. Note that a collection version (which captures the state of a single resource) is very different from a collection baseline (which captures the state of a set of resources).

Baseline-Controlled Collection

A "baseline-controlled collection" is a collection from which baselines can be created (see Section 12).

Version-Controlled Configuration Resource

A "version-controlled configuration resource", or simply "version-controlled configuration", is a special kind of version-controlled resource that is associated with a baseline-controlled collection, and is used to create and access baselines of that collection (see Section 12). When a collection is both version-controlled and baseline-controlled, a client can create a new version of the collection by checking out and checking in that collection, and it can create a new baseline of that collection by checking out and checking in the version-controlled configuration of that collection.

Activity Resource

An "activity resource", or simply "activity", is a resource that selects a set of versions that correspond to a single logical change, where the versions selected from a given version history form a single line of descent through that version history (see Section 13).

11 Merge Feature

When a user wants to accept the changes (new versions) created by someone else, it is important not just to update the version-controlled resources in the user's workspace with those new versions, since this could result in "backing out" changes the user has made to those version-controlled resources. Instead, the versions created in another workspace should be "merged" into the user's version-controlled resources.

The version history of a version-controlled resource provides the information needed to determine the result of the merge. In particular, the merge should select whichever version is later in the line of descent from the root version. In case the versions to be merged are on different lines of descent (neither version is a descendant of the other), neither version should be selected, but instead, a new version should be created that contains the logical merge of the content and dead properties of those versions. The MERGE request can be used to check out each version-controlled resource that requires such a merge, and set the DAV:merge-set property of each checked-out resource to identify the version to be merged. The user is responsible for modifying the content and dead properties of the checked-out resource so that it represents the logical merge of that version, and then adding that version to the DAV:predecessor-set of the checked-out resource.

If the server is capable of automatically performing the merge, it MAY update the content, dead properties, and DAV:predecessor-set of the checked-out resource itself. Before checking in the automatically merged resource, the user is responsible for verifying that the automatic merge is correct.

11.1 Additional Checked-Out Resource Properties

The merge feature introduces the following REQUIRED properties for a checked-out resource.

11.1.1 DAV:merge-set

This property identifies each version that is to be merged into this checked-out resource.

```
<!ELEMENT merge-set (href*)>
```

11.1.2 DAV:auto-merge-set

This property identifies each version that the server has merged into this checked-out resource. The client should confirm that the merge has been performed correctly before moving a URL from the DAV:auto-merge-set to the DAV:predecessor-set of a checked-out resource.

```
<!ELEMENT auto-merge-set (href*)>
```

11.2 MERGE Method

The MERGE method performs the logical merge of a specified version (the "merge source") into a specified version-controlled resource (the "merge target"). If the merge source is neither an ancestor nor a descendant of the DAV:checked-in or DAV:checked-out version of the merge target, the MERGE checks out the merge target (if it is not already checked out) and adds the URL of the merge source to the DAV:merge-set of the merge target. It is then the client's responsibility to update the content and dead properties of the checked-out merge target so that it reflects the logical merge of the merge source into the current state of the merge target. The client indicates that it has completed the update of the merge target, by deleting the merge source URL from the DAV:merge-set of the checked-out merge target, and adding it to the DAV:predecessor-set. As an error check for a client forgetting to complete a merge, the server MUST fail an attempt to CHECKIN a version-controlled resource with a non-empty DAV:merge-set.

When a server has the ability to automatically update the content and dead properties of the merge target to reflect the logical merge of the merge source, it may do so unless DAV:no-auto-merge is specified in the MERGE request body. In order to notify the client that a merge source has been automatically merged, the MERGE request MUST add the URL of the auto-merged source to the DAV:auto-merge-set property of the merge target, and not to the DAV:merge-set property. The client indicates that it has verified that the auto-merge is valid, by deleting the merge source URL from the DAV:auto-merge-set, and adding it to the DAV:predecessor-set.

Multiple merge sources can be specified in a single MERGE request. The set of merge sources for a MERGE request is determined from the DAV:source element of the MERGE request body as follows:

- If DAV:source identifies a version, that version is a merge source.
- If DAV:source identifies a version-controlled resource, the DAV:checked-in version of that version-controlled resource is a merge source.

- If DAV:source identifies a collection, the DAV:checked-in version of each version-controlled resource that is a member of that collection is a merge source.

The request-URL identifies the set of possible merge targets. If the request-URL identifies a collection, any member of the configuration rooted at the request-URL is a possible merge target. The merge target of a particular merge source is the version-controlled or checked-out resource whose DAV:checked-in or DAV:checked-out version is from the same version history as the merge source. If a merge source has no merge target, that merge source is ignored.

The MERGE response identifies the resources that a client must modify to complete the merge. It also identifies the resources modified by the request, so that a client can efficiently update any cached state it is maintaining.

Marshalling:

The request body MUST be a DAV:merge element.

The set of merge sources is determined by the DAV:source element in the request body.

<!ELEMENT merge ANY>

ANY value: A sequence of elements with one DAV:source element, at most one DAV:no-auto-merge element, at most one DAV:no-checkout element, at most one DAV:prop element, and any legal set of elements that can occur in a DAV:checkout element.

<!ELEMENT source (href+)>

<!ELEMENT no-auto-merge EMPTY>

<!ELEMENT no-checkout EMPTY>

prop: see RFC 2518, Section 12.11

The response for a successful request MUST be a 207 Multi-Status, where the DAV:multistatus XML element in the response body identifies all resources that have been modified by the request.

multistatus: see RFC 2518, Section 12.9

The response to a successful request MUST include a Location header containing the URL for the new version created by the checkin.

The response MUST include a Cache-Control:no-cache header.

Preconditions:

(DAV:cannot-merge-checked-out-resource): The DAV:source element MUST NOT identify a checked-out resource. If the DAV:source element identifies a collection, the collection MUST NOT have a member that is a checked-out resource.

(DAV:checkout-not-allowed): If DAV:no-checkout is specified in the request body, it MUST be possible to perform the merge without checking out any of the merge targets.

All preconditions of the CHECKOUT operation apply to the checkouts performed by the request.

Postconditions:

(DAV:ancestor-version): If a merge target is a version-controlled or checked-out resource whose DAV:checked-in version or DAV:checked-out version is the merge source or is a descendant of the merge source, the merge target MUST NOT have been modified by the MERGE.

(DAV:descendant-version): If the merge target was a checked-in version-controlled resource whose DAV:checked-in version was an ancestor of the merge source, an UPDATE operation MUST have been applied to the merge target to set its content and dead properties to be those of the merge source. If the UPDATE method is not supported, the merge target MUST have been checked out, the content and dead properties of the merge target MUST have been set to those of the merge source, and the merge source MUST have been added to the DAV:auto-merge-set of the merge target. The merge target MUST appear in a DAV:response XML element in the response body.

(DAV:checked-out-for-merge): If the merge target was a checked-in version-controlled resource whose DAV:checked-in version was neither a descendant nor an ancestor of the merge source, a CHECKOUT MUST have been applied to the merge target. All XML elements in the DAV:merge XML element that could appear in a DAV:checkout XML element MUST have been used as arguments to the CHECKOUT request. The merge target MUST appear in a DAV:response XML element in the response body.

(DAV:update-merge-set): If the DAV:checked-out version of the merge target is neither equal to nor a descendant of the merge source, the merge source MUST be added to either the DAV:merge-set or the DAV:auto-merge-set of the merge target. The merge target MUST appear in a DAV:response XML element in the response body.

If a merge source has been added to the DAV:auto-merge-set, the content and dead properties of the merge target MUST have been modified by the server to reflect the result of a logical merge of the merge source and the merge target. If a merge source has been added to the DAV:merge-set, the content and dead properties of the merge target MUST NOT have been modified by the server. If DAV:no-auto-merge is specified in the request body, the merge source MUST NOT have been added to the DAV:auto-merge-set.

(DAV:report-properties): If DAV:prop is specified in the request body, the properties specified in the DAV:prop element MUST be reported in the DAV:response elements in the response body.

11.2.1 Example - MERGE

>>REQUEST

```
MERGE /ws/public HTTP/1.1
Host: www.webdav.org
Content-type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:merge xmlns:D="DAV:">
  <D:source>
    <D:href>http://www.webdav.org/ws/dev/sally</D:href>
  </D:source>
</D:merge>
```

>>RESPONSE

```
HTTP/1.1 207 Multi-Status
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx
Cache-Control: no-cache

<?xml version="1.0" encoding="utf-8" ?>
<D:multistatus xmlns:D="DAV:">
  <D:response>
<D:href>http://www.webdav.org/ws/public/src/parse.c</D:href>
<D:status>HTTP/1.1 200 OK</D:status>
  </D:response>
  <D:response>
<D:href>http://www.webdav.org/ws/public/doc/parse.html</D:href>
<D:status>HTTP/1.1 200 OK</D:status>
  </D:response>
</D:multistatus>
```

In this example, the DAV:checked-in versions from the workspace `http://www.webdav.org/ws/dev/sally` are merged into the version-controlled resources in the workspace `http://www.webdav.org/ws/public`. The resources `/ws/public/src/parse.c` and `/ws/public/doc/parse.html` were modified by the request.

11.3 DAV:merge-preview Report

A merge preview describes the changes that would result if the versions specified by the DAV:source element in the request body were to be merged into the resource identified by the request-URL (commonly, a collection).

Marshalling:

The request body MUST be a DAV:merge-preview XML element.

```
<!ELEMENT merge-preview (source)>
<!ELEMENT source (href)>
```

The response body for a successful request MUST be a DAV:merge-preview-report XML element.

```
<!ELEMENT merge-preview-report
  (update-preview | conflict-preview | ignore-preview)*>
```

A DAV:update-preview element identifies a merge target whose DAV:checked-in property would change as a result of the MERGE, and identifies the merge source for that merge target.

```
<!ELEMENT update-preview (target, version)>
<!ELEMENT target (href)>
<!ELEMENT version (href)>
```

A DAV:conflict-preview element identifies a merge target that requires a merge.

```
<!ELEMENT conflict-preview (target, common-ancestor, version)>
```

A DAV:common-ancestor element identifies the version that is a common ancestor of both the merge source and the DAV:checked-in or DAV:checked-out version of the merge target.

```
<!ELEMENT common-ancestor (href)>
```

A DAV:ignore-preview element identifies a version that has no merge target and therefore would be ignored by the merge.

<!ELEMENT ignore-preview (version)>

11.3.1 Example - DAV:merge-preview Report

>>REQUEST

```
REPORT /ws/public HTTP/1.1
Host: www.webdav.org
Content-type: text/xml; charset="utf-8"
Content-Length: xxxx
```

```
<?xml version="1.0" encoding="utf-8" ?>
<D:merge-preview xmlns:D="DAV:">
  <D:source>
    <D:href>http://www.webdav.org/ws/dev/fred</D:href>
  </D:source>
</D:merge-preview>
```

>>RESPONSE

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

```
<?xml version="1.0" encoding="utf-8" ?>
<D:merge-preview-report xmlns:D="DAV:">
  <D:conflict-preview>
    <D:target>
      <D:href>http://www.webdav.org/ws/public/foo.html</D:href>
    </D:target>
    <D:common-ancestor>
      <D:href>http://repo.webdav.org/his/23/ver/18</D:href>
    </D:common-ancestor>
    <D:version>
      <D:href>http://repo.webdav.org/his/23/ver/42</D:href>
    </D:version>
  </D:conflict-preview>
  <D:update-preview>
    <D:target>
      <D:href>http://www.webdav.org/ws/public/bar.html</D:href>
    </D:target>
    <D:version>
      <D:href>http://www.repo/his/42/ver/3</D:href>
    </D:version>
  </D:update-preview>
</D:merge-preview-report>
```

In this example, the merge preview report indicates that version /his/23/ver/42 would be merged in /ws/public/foo.html, and version /his/42/ver/3 would update /ws/public/bar.html if the workspace <http://www.webdav.org/ws/dev/fred> was merged into the workspace <http://www.webdav.org/ws/public>.

11.4 Additional OPTIONS Semantics

If the server supports the merge feature, it MUST include "merge" as a field in the DAV response header from an OPTIONS request on any resource that supports any versioning properties, reports, or methods.

11.5 Additional DELETE Semantics

Additional Postconditions:

(DAV:delete-version-reference): If a version is deleted, any reference to that version in a DAV:merge-set or DAV:auto-merge-set property MUST be removed.

11.6 Additional CHECKIN Semantics

Additional Preconditions:

(DAV:merge-must-be-complete): The DAV:merge-set and DAV:auto-merge-set of the checked-out resource MUST be empty or not exist.

12 Baseline Feature

A configuration is a set of resources that consists of a root collection and all members of that root collection except those resources that are members of another configuration. A configuration that contains a large number of resources can consume a large amount of space on a server. This can make it prohibitively expensive to remember the state of an existing configuration by creating a Depth:infinity copy of its root collection.

A baseline is a version resource that captures the state of each version-controlled member of a configuration. A baseline history is a version history whose versions are baselines. New baselines are created by checking out and then checking in a special kind of version-controlled resource called a version-controlled configuration.

A collection that is under baseline control is called a baseline-controlled collection. In order to allow efficient baseline implementation, the state of a baseline of a collection is limited to

be a set of versions and their names relative to the collection, and the operations on a baseline are limited to the creation of a baseline from a collection, and restoring or merging the baseline back into a collection. A server MAY automatically put a collection under baseline control when it is created, or a client can use the BASELINE-CONTROL method to put a specified collection under baseline control.

As a configuration gets large, it is often useful to break it up into a set of smaller configurations that form the logical "components" of that configuration. In order to capture the fact that a baseline of a configuration is logically extended by a component configuration baseline, the component configuration baseline is captured as a "subbaseline" of the baseline.

The root collection of a configuration is unconstrained with respect to its relationship to the root collection of any of its components. In particular, the root collection of a configuration can have a member that is the root collection of one of its components (e.g., configuration /sys/x can have a component /sys/x/foo), can be a member of the root collection of one of its components (e.g., configuration /sys/y/z can have a component /sys/y), or neither (e.g., configuration /sys/x can have a component /comp/bar).

12.1 Version-Controlled Configuration Properties

Since a version-controlled configuration is a version-controlled resource, it has all the properties of a version-controlled resource. In addition, the baseline feature introduces the following REQUIRED property for a version-controlled configuration.

12.1.1 DAV:baseline-controlled-collection (protected)

This property identifies the collection that contains the version-controlled resources whose DAV:checked-in versions are being tracked by this version-controlled configuration. The DAV:version-controlled-configuration of the DAV:baseline-controlled-collection of a version-controlled configuration MUST identify that version-controlled configuration.

```
<!ELEMENT baseline-controlled-collection (href)>
```

12.2 Checked-Out Configuration Properties

Since a checked-out configuration is a checked-out resource, it has all the properties of a checked-out resource. In addition, the baseline feature introduces the following REQUIRED property for a checked-out configuration.

12.2.1 DAV:subbaseline-set

This property determines the DAV:subbaseline-set property of the baseline that results from checking in this resource.

A server MAY reject attempts to modify the DAV:subbaseline-set of a checked-out configuration.

```
<!ELEMENT subbaseline-set (href*)>
```

12.3 Baseline Properties

The DAV:resourcetype of a baseline MUST be DAV:baseline. Since a baseline is a version resource, it has all the properties of a version resource. In addition, the baseline feature introduces the following REQUIRED properties for a baseline.

12.3.1 DAV:baseline-collection (protected)

This property contains a server-defined URL for a collection, where each member of this collection MUST either be a version-controlled resource with the same DAV:checked-in version and relative name as a version-controlled member of the baseline-controlled collection at the time the baseline was created, or be a collection needed to provide the relative name for a version-controlled resource.

```
<!ELEMENT baseline-collection (href)>
```

12.3.2 DAV:subbaseline-set (protected)

The URLs in the DAV:subbaseline-set property MUST identify a set of other baselines. The subbaselines of a baseline are the baselines identified by its DAV:subbaseline-set and all subbaselines of the baselines identified by its DAV:subbaseline-set.

```
<!ELEMENT subbaseline-set (href*)>
```

12.4 Additional Resource Properties

The baseline feature introduces the following REQUIRED property for a resource.

12.4.1 DAV:version-controlled-configuration (computed)

If the resource is a member of a version-controlled configuration (i.e. the resource is a collection under baseline control or is a member of a collection under baseline control), this property identifies that version-controlled configuration.

<!ELEMENT version-controlled-configuration (href)>

12.5 Additional Workspace Properties

The baseline feature introduces the following REQUIRED property for a workspace.

12.5.1 DAV:baseline-controlled-collection-set (computed)

This property identifies each member of the workspace that is a collection under baseline control (as well as the workspace itself, if it is under baseline control).

<!ELEMENT baseline-controlled-collection-set (href*)>

12.6 BASELINE-CONTROL Method

A collection can be placed under baseline control with a BASELINE-CONTROL request. When a collection is placed under baseline control, the DAV:version-controlled-configuration property of the collection is set to identify a new version-controlled configuration. This version-controlled configuration can be checked out and then checked in to create a new baseline for that collection.

If a baseline is specified in the request body, the DAV:checked-in version of the new version-controlled configuration will be that baseline, and the collection is initialized to contain version-controlled members whose DAV:checked-in versions and relative names are determined by the specified baseline.

If no baseline is specified, a new baseline history is created containing a baseline that captures the state of the version-controlled members of the collection, and the DAV:checked-in version of the version-controlled configuration will be that baseline.

Marshalling:

If a request body is included, it MUST be a DAV:baseline-control XML element.

<!ELEMENT baseline-control ANY>

ANY value: A sequence of elements with at most one DAV:baseline element.

<!ELEMENT baseline (href)>

If a response body for a successful request is included, it MUST be a DAV:baseline-control-response XML element.

<!ELEMENT baseline-control-response ANY>

The response MUST include a Cache-Control:no-cache header.

Preconditions:

(DAV:version-controlled-configuration-must-not-exist): The DAV:version-controlled-configuration property of the collection identified by the request-URL MUST not exist.

(DAV:must-be-baseline): The DAV:href of the DAV:baseline element in the request body MUST identify a baseline.

(DAV:must-have-no-version-controlled-members): If a DAV:baseline element is specified in the request body, the collection identified by the request-URL MUST have no version-controlled members.

(DAV:one-baseline-controlled-collection-per-history-per-workspace): If the request-URL identifies a workspace or a member of a workspace, and if a baseline is specified in a DAV:baseline element in the request body, then there MUST NOT be another collection in that workspace whose DAV:version-controlled-configuration property identifies a version-controlled configuration for the baseline history of that baseline.

Postconditions:

(DAV:create-version-controlled-configuration): A new version-controlled configuration is created, whose DAV:baseline-controlled-collection property identifies the collection.

(DAV:reference-version-controlled-configuration): The DAV:version-controlled-configuration of the collection identifies the new version-controlled configuration.

(DAV:select-existing-baseline): If the request body specifies a baseline, the DAV:checked-in property of the new version-controlled configuration MUST have been set to identify this baseline. A version-controlled member of the collection will be created for each version in the baseline, where the version-controlled member will have the content and dead properties of that version, and will have the same name relative to the collection as the corresponding version-controlled resource had when the baseline was created. Any nested collections that are needed to provide the appropriate name for a version-controlled member will be created.

(DAV:create-new-baseline): If no baseline is specified in the request body, the request MUST have created a new baseline history at a server-defined URL, and MUST have created a new baseline in that baseline history. The DAV:baseline-collection of the new baseline MUST identify a collection whose members have the same relative name and DAV:checked-in version as the version-controlled members of the request collection. The DAV:checked-in property of the new version-controlled configuration MUST identify the new baseline.

12.6.1 Example - BASELINE-CONTROL

>>REQUEST

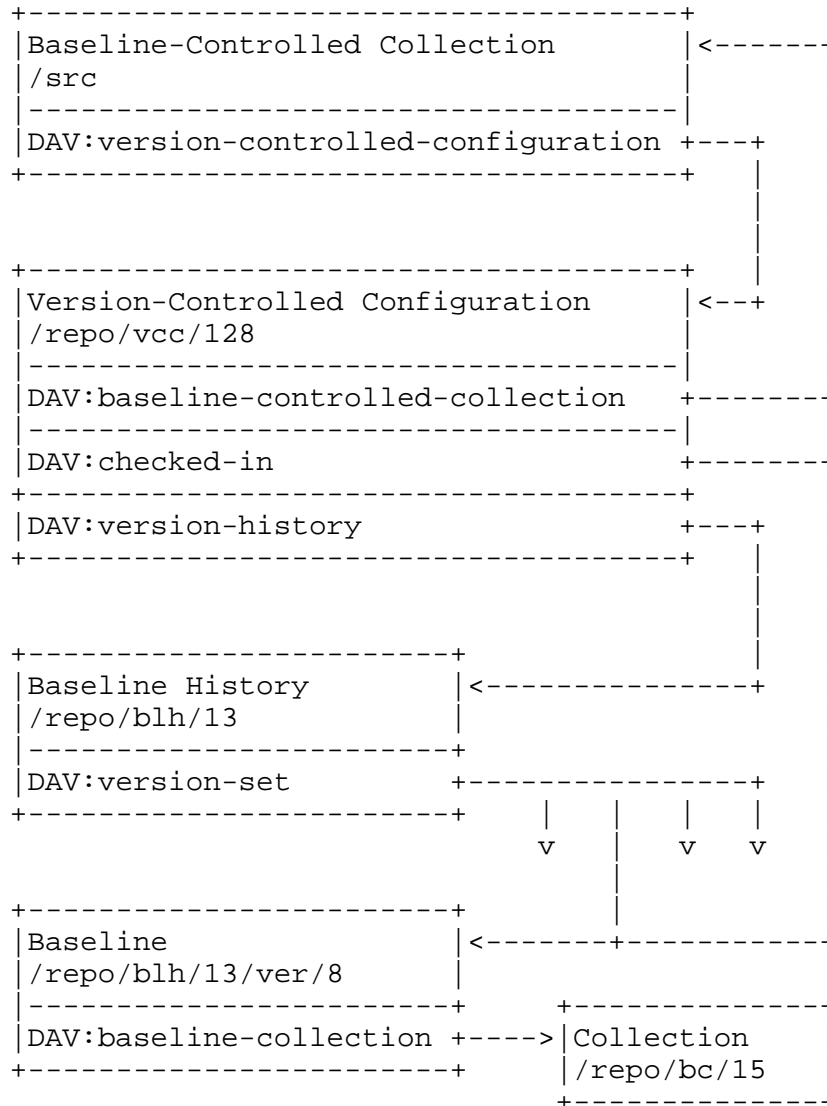
```
BASELINE-CONTROL /src HTTP/1.1
Host: www.webdav.org
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx

<?xml version="1.0" encoding="utf-8" ?>
<D:baseline-control xmlns:D="DAV:">
  <D:href>http://www.webdav.org/repo/blh/13/ver/8</D:href>
</D:baseline-control>
```

>>RESPONSE

```
HTTP/1.1 200 OK
Cache-Control: no-cache
Content-Length: 0
```

In this example, the collection /src is placed under baseline control, and is populated with members from an existing baseline. A new version-controlled configuration (/repo/vcc/128) is created and associated with /src, and /src is initialized with version-controlled members whose DAV:checked-in versions are those selected by the DAV:baseline-collection (/repo/bc/15) of the specified baseline (/repo/blh/13/ver/8). The following diagram illustrates the resulting state on the server.



In order to create new baselines of /src, /repo/vcc/128 can be checked out, new versions can be created or selected by the version-controlled members of /src, and then /repo/vcc/128 can be checked in to capture the current state of those version-controlled members.

12.7 DAV:compare-baseline Report

A DAV:compare-baseline report contains the differences between the baseline identified by the request-URL (the "request baseline") and the baseline specified in the request body (the "compare baseline").

Marshalling:

The request body MUST be a DAV:compare-baseline XML element.

```
<!ELEMENT compare-baseline (href)>
```

The response body for a successful request MUST be a DAV:compare-baseline-report XML element.

```
<!ELEMENT compare-baseline-report  
  (added-version | deleted-version | changed-version)*>
```

A DAV:added-version element identifies a version that is the DAV:checked-in version of a member of the DAV:baseline-collection of the compare baseline, but no version in the version history of that version is the DAV:checked-in version of a member of the DAV:baseline-collection of the request baseline.

```
<!ELEMENT added-version (href)>
```

A DAV:deleted-version element identifies a version that is the DAV:checked-in version of a member of the DAV:baseline-collection of the request baseline, but no version in the version history of that version is the DAV:checked-in version of a member of the DAV:baseline-collection of the compare baseline.

```
<!ELEMENT deleted-version (href)>
```

A DAV:changed-version element identifies two different versions from the same version history that are the DAV:checked-in version of the DAV:baseline-collection of the request baseline and the compare baseline, respectively.

```
<!ELEMENT changed-version (href, href)>
```

Preconditions:

(DAV:must-be-baseline): The DAV:href in the request body MUST identify a baseline.

(DAV:baselines-from-same-history): A server MAY require that the baselines being compared be from the same baseline history.

12.7.1 Example - DAV:compare-baseline Report

>>REQUEST

```
REPORT /bl-his/12/bl/14 HTTP/1.1
Host: repo.webdav.com
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx
```

```
<?xml version="1.0" encoding="utf-8" ?>
<D:compare-baseline xmlns:D="DAV:">
  <D:href>http://repo.webdav.org/bl-his/12/bl/15</D:href>
</D:compare-baseline>
```

>>RESPONSE

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

```
<?xml version="1.0" encoding="utf-8" ?>
<D:compare-baseline-report xmlns:D="DAV:">
  <D:added-version>
    <D:href>http://repo.webdav.org/his/23/ver/8</D:href>
  </D:added-version>
  <D:changed-version>
    <D:href>http://repo.webdav.org/his/29/ver/12</D:href>
    <D:href>http://repo.webdav.org/his/29/ver/19</D:href>
  </D:changed-version>
  <D:deleted-version>
    <D:href>http://repo.webdav.org/his/12/ver/4</D:href>
  </D:deleted-version>
</D:compare-baseline-report>
```

In this example, the differences between baseline 14 and baseline 15 of <http://repo.webdav.org/bl-his/12> are identified.

12.8 Additional OPTIONS Semantics

If a server supports the baseline feature, it MUST include "baseline" as a field in the DAV response header from an OPTIONS request on any resource that supports any versioning properties, reports, or methods.

12.9 Additional MKCOL Semantics

Additional Postconditions:

If a server automatically puts a newly created collection under baseline control, all postconditions for BASELINE-CONTROL apply to the MKCOL.

12.10 Additional COPY Semantics

Additional Postconditions:

If the request creates a new collection at the Destination, and a server automatically puts a newly created collection under baseline control, all postconditions for BASELINE-CONTROL apply to the COPY.

12.11 Additional CHECKOUT Semantics

Additional Preconditions:

(DAV:must-not-update-baseline-collection): If the request-URL identifies a member of the configuration rooted at the DAV:baseline-collection of a baseline, the request MUST fail.

12.12 Additional CHECKIN Semantics

Additional Preconditions:

(DAV:no-checked-out-baseline-controlled-collection-members): If the request-URL identifies a version-controlled configuration, all version-controlled members of the DAV:baseline-controlled-collection of the version-controlled configuration MUST be checked-in.

(DAV:one-version-per-history-per-baseline): If the request-URL identifies a version-controlled configuration, the set of versions selected by that version-controlled configuration MUST contain at most one version from any version history, where a version is selected by a version-controlled configuration if the version is identified by the DAV:checked-in property of any member of the

configuration rooted at the DAV:baseline-controlled collection of that version-controlled configuration, or is identified by the DAV:checked-in property of any member of the configuration rooted at the DAV:baseline-collection of any subbaseline of that version-controlled configuration.

(DAV:cannot-modify-version-controlled-configuration): If the request-URL identifies a version-controlled member of a baseline-controlled collection whose version-controlled configuration is checked-in, the request MUST fail unless the DAV:auto-version property of the version-controlled configuration will automatically check out that version-controlled configuration when it is modified.

Additional Postconditions:

(DAV:create-baseline-collection): If the request-URL identifies a version-controlled configuration, the DAV:baseline-collection of the new baseline identifies a collection whose members have the same relative name and DAV:checked-in version as the members of the DAV:baseline-controlled-collection of the version-controlled configuration at the time of the request.

(DAV:modify-configuration): If the request-URL identifies a version-controlled member of a baseline-controlled collection, this is a modification to the version-controlled configuration of that baseline-controlled collection, and standard auto-versioning semantics apply.

12.13 Additional UPDATE Semantics

Additional Preconditions:

(DAV:baseline-controlled-members-must-be-checked-in): If the request-URL identifies a version-controlled configuration, then all version-controlled members of the DAV:baseline-controlled-collection of that version-controlled configuration MUST be checked-in.

(DAV:must-not-update-baseline-collection): If the request-URL identifies a member of the configuration rooted at the DAV:baseline-collection of a baseline, the request MUST fail.

(DAV:cannot-modify-version-controlled-configuration): If the request updates the DAV:checked-in property of any version-controlled member of a baseline-controlled collection whose version-controlled configuration is checked-in, the request MUST

fail unless the DAV:auto-version property of the version-controlled configuration will automatically check out that version-controlled configuration when it is modified.

Additional Postconditions:

(DAV:set-baseline-controlled-collection-members): If the request updated the DAV:checked-in property of a version-controlled configuration, then the version-controlled members of the DAV:baseline-controlled-collection of that version-controlled configuration MUST have been updated so that they have the same relative name, content, and dead properties as the members of the DAV:baseline-collection of the baseline. In particular:

- A version-controlled member for a given version history MUST have been deleted if there is no version-controlled member for that version history in the DAV:baseline-collection of the baseline.
- A version-controlled member for a given version history MUST have been renamed if its name relative to the baseline-controlled collection is different from that of the version-controlled member for that version history in the DAV:baseline-collection of the baseline.
- A new version-controlled member MUST have been created for each member of the DAV:baseline-collection of the baseline for which there is no corresponding version-controlled member in the baseline-controlled collection.
- An UPDATE request MUST have been applied to each version-controlled member for a given version history whose DAV:checked-in version is not the same as that of the version-controlled member for that version history in the DAV:baseline-collection of the baseline.

(DAV:update-subbaselines): If the request updated a version-controlled configuration whose DAV:baseline-controlled-collection contains a baseline-controlled member for one of the subbaselines of the request baseline, then the DAV:checked-in property of the version-controlled configuration of that baseline-controlled member MUST have been updated to be that subbaseline. If the request updated a version-controlled configuration whose DAV:baseline-controlled-collection is a member of a workspace that contains a baseline-controlled member for one of the subbaselines of the request baseline, then the DAV:checked-in property of the version-controlled configuration of that baseline-controlled member MUST have been updated to be that subbaseline.

(DAV:modify-configuration): If the request updated the DAV:checked-in property of any version-controlled member of a baseline-controlled collection, and if this DAV:checked-in property differs from the DAV:checked-in property of the corresponding version-controlled member of the DAV:baseline-collection of the DAV:checked-in baseline of the DAV:version-controlled-configuration of the baseline-controlled collection, then this is a modification to that version-controlled configuration, and standard auto-versioning semantics apply.

12.14 Additional MERGE Semantics

If the merge source is a baseline, the merge target is a version-controlled configuration for the baseline history of that baseline, where the baseline-controlled collection of that version-controlled configuration is a member of the collection identified by the request-URL.

Additional Preconditions:

(DAV:must-not-update-baseline-collection): Same semantics as UPDATE (see Section 12.13).

(DAV:cannot-modify-version-controlled-configuration): Same semantics as UPDATE (see Section 12.13).

Additional Postconditions:

(DAV:merge-baseline): If the merge target is a version-controlled configuration whose DAV:checked-out baseline is not a descendant of the merge baseline, then the merge baseline MUST have been added to the DAV:auto-merge-set of a version-controlled configuration. The DAV:checked-in version of each member of the DAV:baseline-collection of that baseline MUST have been merged into the DAV:baseline-controlled-collection of that version-controlled configuration.

(DAV:merge-subbaselines): If the merge target is a version-controlled configuration whose DAV:baseline-controlled-collection contains a baseline-controlled member for one of the subbaselines of the merge baseline, then that subbaseline MUST have been merged into the version-controlled configuration of that baseline-controlled member. If the merge target is a version-controlled configuration whose DAV:baseline-controlled-collection is a member of a workspace that contains a baseline-controlled member for one of the subbaselines of the merge baseline, then that subbaseline MUST have been merged into the version-controlled configuration of that baseline-controlled member.

(DAV:set-baseline-controlled-collection-members): Same semantics as UPDATE (see Section 12.13).

(DAV:modify-configuration): Same semantics as UPDATE (see Section 12.13).

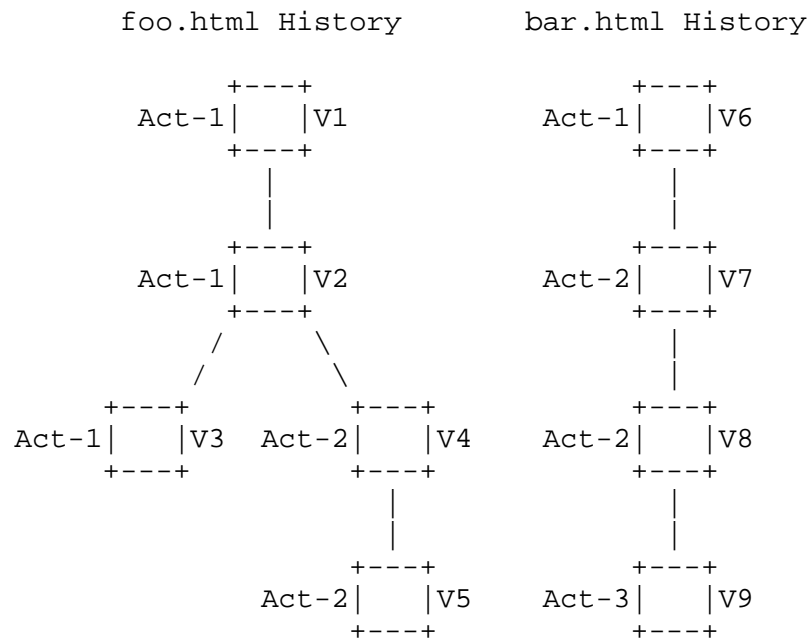
13 Activity Feature

An activity is a resource that selects a set of versions that are on a single "line of descent", where a line of descent is a sequence of versions connected by successor relationships. If an activity selects versions from multiple version histories, the versions selected in each version history must be on a single line of descent.

A common problem that motivates the use of activities is that it is often desirable to perform several different logical changes in a single workspace, and then selectively merge a subset of those logical changes to other workspaces. An activity can be used to represent a single logical change, where an activity tracks all the resources that were modified to effect that single logical change. When a version-controlled resource is checked out, the user specifies which activity should be associated with a new version that will be created when that version-controlled resource is checked in. It is then possible to select a particular logical change for merging into another workspace, by specifying the appropriate activity in a MERGE request.

Another common problem is that although a version-controlled resource may need to have multiple lines of descent, all work done by members of a given team must be on a single line of descent (to avoid merging between team members). An activity resource provides the mechanism for addressing this problem. When a version-controlled resource is checked out, a client can request that an existing activity be used or that a new activity be created. Activity semantics then ensure that all versions in a given version history that are associated with an activity are on a single line of descent. If all members of a team share a common activity (or sub-activities of a common activity), then all changes made by members of that team will be on a single line of descent.

The following diagram illustrates activities. Version V5 is the latest version of foo.html selected by activity Act-2, and version V8 is the latest version of bar.html selected by activity Act-2.



Activities appear under a variety of names in existing versioning systems. When an activity is used to capture a logical change, it is commonly called a "change set". When an activity is used to capture a line of descent, it is commonly called a "branch". When a system supports both branches and change sets, it is often useful to require that a particular change set occur on a particular branch. This relationship can be captured by making the change set activity be a "subactivity" of the branch activity.

13.1 Activity Properties

The DAV:resourcetype of an activity MUST be DAV:activity.

The activity feature introduces the following REQUIRED properties for an activity.

13.1.1 DAV:activity-version-set (computed)

This property identifies each version whose DAV:activity-set property identifies this activity. Multiple versions of a single version history can be selected by an activity's DAV:activity-version-set

property, but all DAV:activity-version-set versions from a given version history must be on a single line of descent from the root version of that version history.

```
<!ELEMENT activity-version-set (href*)>
```

13.1.2 DAV:activity-checkout-set (computed)

This property identifies each checked-out resource whose DAV:activity-set identifies this activity.

```
<!ELEMENT activity-checkout-set (href*)>
```

13.1.3 DAV:subactivity-set

This property identifies each activity that forms a part of the logical change being captured by this activity. An activity behaves as if its DAV:activity-version-set is extended by the DAV:activity-version-set of each activity identified in the DAV:subactivity-set. In particular, the versions in this extended set MUST be on a single line of descent, and when an activity selects a version for merging, the latest version in this extended set is the one that will be merged.

A server MAY reject attempts to modify the DAV:subactivity-set of an activity.

```
<!ELEMENT subactivity-set (href*)>
```

13.1.4 DAV:current-workspace-set (computed)

This property identifies each workspace whose DAV:current-activity-set identifies this activity.

```
<!ELEMENT current-workspace-set (href*)>
```

13.2 Additional Version Properties

The activity feature introduces the following REQUIRED property for a version.

13.2.1 DAV:activity-set

This property identifies the activities that determine to which logical changes this version contributes, and on which lines of descent this version appears. A server MAY restrict the DAV:activity-set to identify a single activity. A server MAY refuse to allow the value of the DAV:activity-set property of a version to be modified.

```
<!ELEMENT activity-set (href*)>
```

13.3 Additional Checked-Out Resource Properties

The activity feature introduces the following REQUIRED properties for a checked-out resource.

13.3.1 DAV:unreserved

This property of a checked-out resource indicates whether the DAV:activity-set of another checked-out resource associated with the version history of this version-controlled resource can have an activity that is in the DAV:activity-set property of this checked-out resource.

A result of the requirement that an activity must form a single line of descent through a given version history is that if multiple checked-out resources for a given version history are checked out unreserved into a single activity, only the first CHECKIN will succeed. Before another of these checked-out resources can be checked in, the user will first have to merge into that checked-out resource the latest version selected by that activity from that version history, and then modify the DAV:predecessor-set of that checked-out resource to identify that version.

```
<!ELEMENT unreserved (#PCDATA)>
PCDATA value: boolean
```

13.3.2 DAV:activity-set

This property of a checked-out resource determines the DAV:activity-set property of the version that results from checking in this resource.

13.4 Additional Workspace Properties

The activity feature introduces the following REQUIRED property for a workspace.

13.4.1 DAV:current-activity-set

This property identifies the activities that currently are being performed in this workspace. When a member of this workspace is checked out, if no activity is specified in the checkout request, the DAV:current-activity-set will be used. This allows an activity-unaware client to update a workspace in which activity tracking is required. The DAV:current-activity-set MAY be restricted to identify at most one activity.

```
<!ELEMENT current-activity-set (href*)>
```

13.5 MKACTIVITY Method

A MKACTIVITY request creates a new activity resource. A server MAY restrict activity creation to particular collections, but a client can determine the location of these collections from a DAV:activity-collection-set OPTIONS request.

Marshalling:

If a request body is included, it MUST be a DAV:mkactivity XML element.

```
<!ELEMENT mkactivity ANY>
```

If a response body for a successful request is included, it MUST be a DAV:mkactivity-response XML element.

```
<!ELEMENT mkactivity-response ANY>
```

The response MUST include a Cache-Control:no-cache header.

Preconditions:

(DAV:resource-must-be-null): A resource MUST NOT exist at the request-URL.

(DAV:activity-location-ok): The request-URL MUST identify a location where an activity can be created.

Postconditions:

(DAV:initialize-activity): A new activity exists at the request-URL. The DAV:resourcetype of the activity MUST be DAV:activity.

13.5.1 Example - MKACTIVITY

>>REQUEST

```
MKACTIVITY /act/test-23 HTTP/1.1
Host: repo.webdav.org
Content-Length: 0
```

>>RESPONSE

```
HTTP/1.1 201 Created
Cache-Control: no-cache
```

In this example, a new activity is created at <http://repo.webdav.org/act/test-23>.

13.6 DAV:latest-activity-version Report

The DAV:latest-activity-version report can be applied to a version history to identify the latest version that is selected from that version history by a given activity.

Marshalling:

The request body MUST be a DAV:latest-activity-version XML element.

```
<!ELEMENT latest-activity-version (href)>
```

The response body for a successful request MUST be a DAV:latest-activity-version-report XML element.

```
<!ELEMENT latest-activity-version-report (href)>
```

The DAV:href of the response body MUST identify the version of the given version history that is a member of the DAV:activity-version-set of the given activity and has no descendant that is a member of the DAV:activity-version-set of that activity.

Preconditions:

(DAV:must-be-activity): The DAV:href in the request body MUST identify an activity.

13.7 Additional OPTIONS Semantics

If the server supports the activity feature, it **MUST** include "activity" as a field in the DAV response header from an OPTIONS request on any resource that supports any versioning properties, reports, or methods.

A DAV:activity-collection-set element **MAY** be included in the request body to identify collections that may contain activity resources.

Additional Marshalling:

If an XML request body is included, it **MUST** be a DAV:options XML element.

<!ELEMENT options ANY>

ANY value: A sequence of elements with at most one DAV:activity-collection-set element.

If an XML response body for a successful request is included, it **MUST** be a DAV:options-response XML element.

<!ELEMENT options-response ANY>

ANY value: A sequence of elements with at most one DAV:activity-collection-set element.

<!ELEMENT activity-collection-set (href*)>

If DAV:activity-collection-set is included in the request body, the response body for a successful request **MUST** contain a DAV:activity-collection-set element identifying collections that may contain activities. An identified collection **MAY** be the root collection of a tree of collections, all of which may contain activities. Since different servers can control different parts of the URL namespace, different resources on the same host **MAY** have different DAV:activity-collection-set values. The identified collections **MAY** be located on different hosts from the resource.

13.8 Additional DELETE Semantics

Additional Postconditions:

(DAV:delete-activity-reference): If an activity is deleted, any reference to that activity in a DAV:activity-set, DAV:subactivity-set, or DAV:current-activity-set **MUST** be removed.

13.9 Additional MOVE Semantics

Additional Postconditions:

(DAV:update-checked-out-reference): If a checked-out resource is moved, any reference to that resource in a DAV:activity-checkout-set property MUST be updated to refer to the new location of that resource.

(DAV:update-activity-reference): If the request-URL identifies an activity, any reference to that activity in a DAV:activity-set, DAV:subactivity-set, or DAV:current-activity-set MUST be updated to refer to the new location of that activity.

(DAV:update-workspace-reference): If the request-URL identifies a workspace, any reference to that workspace in a DAV:current-workspace-set property MUST be updated to refer to the new location of that workspace.

13.10 Additional CHECKOUT Semantics

A CHECKOUT request MAY specify the DAV:activity-set for the checked-out resource.

Additional Marshalling:

<!ELEMENT checkout ANY> ANY value: A sequence of elements with at most one DAV:activity-set and at most one DAV:unreserved.

<!ELEMENT activity-set (href+ | new)>
<!ELEMENT new EMPTY>
<!ELEMENT unreserved EMPTY>

Additional Preconditions:

(DAV:one-checkout-per-activity-per-history): If there is a request activity set, unless DAV:unreserved is specified, another checkout from a version of that version history MUST NOT select an activity in that activity set.

(DAV:linear-activity): If there is a request activity set, unless DAV:unreserved is specified, the selected version MUST be a descendant of all other versions of that version history that select that activity.

Additional Postconditions:

(DAV:initialize-activity-set): The DAV:activity-set of the checked-out resource is set as follows:

- If DAV:new is specified as the DAV:activity-set in the request body, then a new activity created by the server is used.
- Otherwise, if activities are specified in the request body, then those activities are used.
- Otherwise, if the version-controlled resource is a member of a workspace and the DAV:current-activity-set of the workspace is set, then those activities are used.
- Otherwise, the DAV:activity-set of the DAV:checked-out version is used.

(DAV:initialize-unreserved): If DAV:unreserved was specified in the request body, then the DAV:unreserved property of the checked-out resource MUST be "true".

13.10.1 Example - CHECKOUT with an activity

>>REQUEST

```
CHECKOUT /ws/public/foo.html HTTP/1.1
Host: www.webdav.org
Content-Type: text/xml; charset="utf-8"
Content-Length: xxxx
```

```
<?xml version="1.0" encoding="utf-8" ?>
<D:checkout xmlns:D="DAV:">
  <D:activity-set>
    <D:href>http://repo.webdav.org/act/fix-bug-23</D:href>
  </D:activity-set>
</D:checkout>
```

>>RESPONSE

```
HTTP/1.1 200 OK
Cache-Control: no-cache
```

In this example, the CHECKOUT is being performed in the `http://repo.webdav.org/act/fix-bug-23` activity.

13.11 Additional CHECKIN Semantics

Additional Preconditions:

(DAV:linear-activity): Any version which is in the version history of the checked-out resource and whose DAV:activity-set identifies an activity from the DAV:activity-set of the checked-out resource MUST be an ancestor of the checked-out resource.

(DAV:atomic-activity-checkin): If the request-URL identifies an activity, the server MAY fail the request if any of the checked-out resources in the DAV:activity-checkout-set of either that activity or any subactivity of that activity cannot be checked in.

Additional Postconditions:

(DAV:initialize-activity-set): The DAV:activity-set of the new version MUST have been initialized to be the same as the DAV:activity-set of the checked-out resource.

(DAV:activity-checkin): If the request-URL identified an activity, the server MUST have successfully applied the CHECKIN request to each checked-out resource in the DAV:activity-checkout-set of both that activity and any subactivity of that activity.

13.12 Additional MERGE Semantics

If the DAV:source element of the request body identifies an activity, then for each version history containing a version selected by that activity, the latest version selected by that activity is a merge source. Note that the versions selected by an activity are the versions in its DAV:activity-version-set unioned with the versions selected by the activities in its DAV:subactivity-set.

Additional Marshalling:

<!ELEMENT checkin-activity EMPTY>

Additional Postconditions:

(DAV:checkin-activity): If DAV:checkin-activity is specified in the request body, and if the DAV:source element in the request body identifies an activity, a CHECKIN request MUST have been successfully applied to that activity before the merge sources were determined.

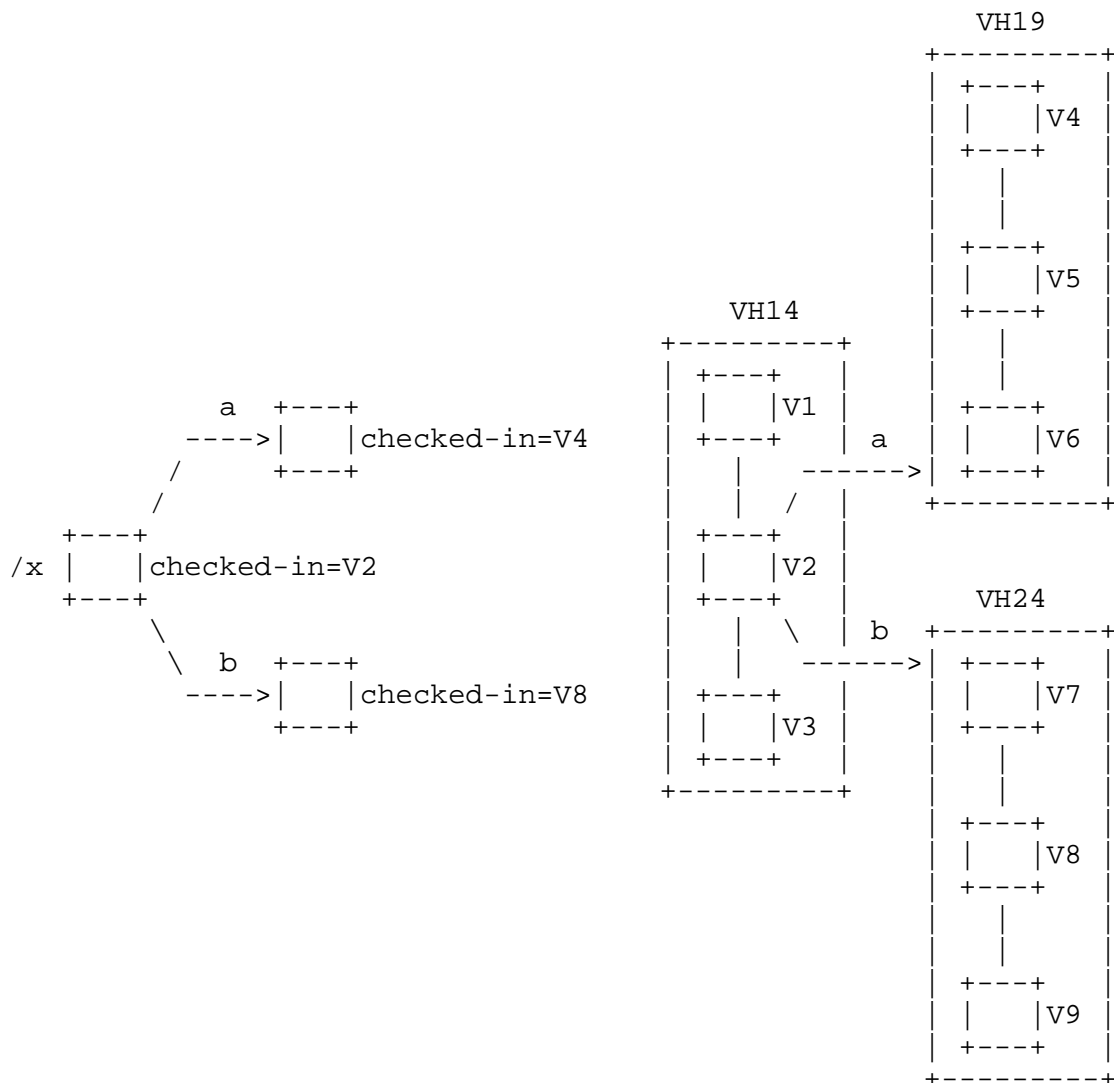
14 Version-Controlled-Collection Feature

As with any versionable resource, when a collection is put under version control, a version history resource is created to contain versions for that version-controlled collection. In order to preserve standard versioning semantics (a version of a collection should not be modifiable), a collection version only records information about the version-controlled bindings of that collection.

In order to cleanly separate a modification to the namespace from a modification to content or dead properties, a version of a collection has no members, but instead records in its DAV:version-controlled-binding-set property the binding name and version history resource of each version-controlled internal member of that collection. If, instead, a collection version contained bindings to other versions, creating a new version of a resource would require creating a new version of all the collection versions that contain that resource, which would cause activities to become entangled. For example, suppose a "feature-12" activity created a new version of /x/y/a.html. If a collection version contained bindings to versions of its members, a new version of /x/y would have to be created to contain the new version of /x/y/a.html, and a new version of /x would have to be created to contain the new version of /x/y. Now suppose a "bugfix-47" activity created a new version of /x/z/b.html. Again, a new version of /x/z and a new version of /x would have to be created to contain the new version of /x/y/b.html. But now it is impossible to merge just "bugfix-47" into another workspace without "feature-12", because the version of /x that contains the desired version of /x/z/b.html also contains version of /x/y/a.html created for "feature-12". If, instead, a collection version just records the binding name and version history resource of each version-controlled internal member, changing the version selected by a member of that collection would not require a new version of the collection. The new version is still in the same version history so no new collection version is required, and "feature-12" and "bugfix-47" would not become entangled.

In the following example, there are three version histories, named VH14, VH19, and VH24, where VH14 contains versions of a collection. The version-controlled collection /x has version V2 of version history VH14 as its DAV:checked-in version. Since V2 has recorded two version controlled bindings, one with binding name "a" to version history VH19, and the other with binding name "b" to version history VH24, /x MUST have two version-controlled bindings, one named "a" to a version-controlled resource for history VH19, and the other named "b" to a version-controlled resource for history VH24. The version-

controlled resource `/x/a` currently has V4 of VH19 as its DAV:checked-in version, while `/x/b` has V8 of VH24 as its DAV:checked-in version.



For any request (e.g., DELETE, MOVE, COPY) that modifies a version-controlled binding of a checked-in version-controlled collection, the request MUST fail unless the version-controlled collection has a DAV:auto-version property that will automatically check out the version-controlled collection when it is modified.

Although a collection version only records the version-controlled bindings of a collection, a version-controlled collection MAY contain both version-controlled and non-version-controlled bindings. Non-

version-controlled bindings are not under version control, and therefore can be added or deleted without checking out the version-controlled collection.

Note that a collection version captures only a defined subset of the state of a collection. In particular, a version of a collection captures its dead properties and its bindings to version-controlled resources, but not its live properties or bindings to non-version-controlled resources.

When a server supports the working-resource feature, a client can check out a collection version to create a working collection. Unlike a version-controlled collection, which contains bindings to version-controlled resources and non-version-controlled resources, a working collection contains bindings to version history resources and non-version-controlled resources. In particular, a working collection is initialized to contain bindings to the version history resources specified by the DAV:version-controlled-binding-set of the checked out collection version. The members of a working collection can then be deleted or moved to another working collection. Non-version-controlled resources can be added to a working collection with methods such as PUT, COPY, and MKCOL. When a working collection is checked in, a VERSION-CONTROL request is automatically applied to every non-version-controlled member of the working collection, and each non-version-controlled member is replaced by its newly created version history. The DAV:version-controlled-binding-set of the new version resulting from checking in a working collection contains the binding name and version history URL for each member of the working collection.

14.1 Version-Controlled Collection Properties

A version-controlled collection has all the properties of a collection and of a version-controlled resource. In addition, the version-controlled-collection feature introduces the following REQUIRED property for a version-controlled collection.

14.1.1 DAV:eclipsed-set (computed)

This property identifies the non-version-controlled internal members of the collection that currently are eclipsing a version-controlled internal member of the collection.

```
!ELEMENT eclipsed-set (binding-name*)>
<!ELEMENT binding-name (#PCDATA)>
PCDATA value: URL segment
```

An UPDATE or MERGE request can give a version-controlled collection a version-controlled internal member that has the same name as an existing non-version-controlled internal member. In this case, the non-version-controlled internal member takes precedence and is said to "eclipse" the new versioned-controlled internal member. If the non-version-controlled internal member is removed (e.g., by a DELETE or MOVE), the version-controlled internal member is exposed.

14.2 Collection Version Properties

A collection version has all the properties of a version. In addition, the version-controlled-collection feature introduces the following REQUIRED property for a collection version.

14.2.1 DAV:version-controlled-binding-set (protected)

This property captures the name and version-history of each version-controlled internal member of a collection.

```
<!ELEMENT version-controlled-binding-set
(version-controlled-binding*)>
<!ELEMENT version-controlled-binding
(binding-name, version-history)>
<!ELEMENT binding-name (#PCDATA)>
PCDATA value: URL segment
<!ELEMENT version-history (href)>
```

14.3 Additional OPTIONS Semantics

If the server supports the version-controlled-collection feature, it MUST include "version-controlled-collection" as a field in the DAV response header from an OPTIONS request on any resource that supports any versioning properties, reports, or methods.

14.4 Additional DELETE Semantics

Additional Preconditions:

(DAV:cannot-modify-checked-in-parent): If the request-URL identifies a version-controlled resource, the DELETE MUST fail when the collection containing the version-controlled resource is a checked-in version-controlled collection, unless DAV:auto-version semantics will automatically check out the version-controlled collection.

14.5 Additional MKCOL Semantics

Additional Preconditions:

If the request creates a new resource that is automatically placed under version control, all preconditions for VERSION-CONTROL apply to the request.

Additional Postconditions:

If the new collection is automatically put under version control, all postconditions for VERSION-CONTROL apply to the request.

14.6 Additional COPY Semantics

Additional Preconditions:

(DAV:cannot-copy-collection-version): If the source of the request is a collection version, the request MUST fail.

14.7 Additional MOVE Semantics

Additional Preconditions:

(DAV:cannot-modify-checked-in-parent): If the source of the request is a version-controlled resource, the request MUST fail when the collection containing the source is a checked-in version-controlled collection, unless DAV:auto-version semantics will automatically check out that version-controlled collection.

(DAV:cannot-modify-destination-checked-in-parent): If the source of the request is a version-controlled resource, the request MUST fail when the collection containing the destination is a checked-in version-controlled collection, unless DAV:auto-version semantics will automatically check out that version-controlled collection.

14.8 Additional VERSION-CONTROL Semantics

Additional Preconditions:

(DAV:cannot-modify-checked-in-parent): If the parent of the request-URL is a checked-in version-controlled collection, the request MUST fail unless DAV:auto-version semantics will automatically check out that version-controlled collection.

Additional Postconditions:

(DAV:new-version-controlled-collection): If the request body identified a collection version, the collection at the request-URL MUST contain a version-controlled internal member for each DAV:version-controlled-binding specified in the DAV:version-controlled-binding-set of the collection version, where the name and DAV:version-history of the internal member MUST be the DAV:binding-name and the DAV:version-history specified by the DAV:version-controlled-binding. If the internal member is a member of a workspace, and there is another member of the workspace for the same version history, those two members MUST identify the same version-controlled resource; otherwise, a VERSION-CONTROL request with a server selected version of the version history MUST have been applied to the URL for that internal member.

14.9 Additional CHECKOUT Semantics

Additional Postconditions:

(DAV:initialize-version-history-bindings): If the request has been applied to a collection version, the new working collection MUST be initialized to contain a binding to each of the history resources identified in the DAV:version-controlled-binding-set of that collection version.

14.10 Additional CHECKIN Semantics

Additional Postconditions:

(DAV:initialize-version-controlled-bindings): If the request-URL identified a version-controlled collection, then the DAV:version-controlled-binding-set of the new collection version MUST contain a DAV:version-controlled-binding that identifies the binding name and version history for each version-controlled binding of the version-controlled collection.

(DAV:version-control-working-collection-members): If the request-URL identified a working collection, a VERSION-CONTROL request MUST have been automatically applied to every non-version-controlled member of the working collection, and each non-version-controlled member MUST have been replaced by its newly created version history. If a working collection member was a non-version-controlled collection, every member of the non-version-controlled collection MUST have been placed under version

control before the non-version-controlled collection was placed under version control. The DAV:version-controlled-binding-set of the new collection version MUST contain a DAV:version-controlled-binding that identifies the binding name and the version history URL for each member of the working collection.

14.11 Additional UPDATE and MERGE Semantics

Additional Postconditions:

(DAV:update-version-controlled-collection-members): If the request modified the DAV:checked-in version of a version-controlled collection, then the version-controlled members of that version-controlled collection MUST have been updated. In particular:

- A version-controlled internal member MUST have been deleted if its version history is not identified by the DAV:version-controlled-binding-set of the new DAV:checked-in version.
- A version-controlled internal member for a given version history MUST have been renamed if its binding name differs from the DAV:binding-name for that version history in the DAV:version-controlled-binding-set of the new DAV:checked-in version.
- A new version-controlled internal member MUST have been created when a version history is identified by the DAV:version-controlled-binding-set of the DAV:checked-in version, but there was no member of the version-controlled collection for that version history. If a new version-controlled member is in a workspace that already has a version-controlled resource for that version history, then the new version-controlled member MUST be just a binding (i.e., another name for) that existing version-controlled resource. Otherwise, the content and dead properties of the new version-controlled member MUST have been initialized to be those of the version specified for that version history by the request. If no version is specified for that version history by the request, the version selected is server defined.

15 Internationalization Considerations

This specification has been designed to be compliant with the IETF Policy on Character Sets and Languages [RFC2277]. Specifically, where human-readable strings exist in the protocol, either their charset is explicitly stated, or XML mechanisms are used to specify the charset used. Additionally, these human-readable strings all have the ability to express the natural language of the string.

Most of the human-readable strings in this protocol appear in properties, such as DAV:creator-displayname. As defined by RFC 2518, properties have their values marshaled as XML. XML has explicit provisions for character set tagging and encoding, and requires that XML processors read XML elements encoded, at minimum, using the UTF-8 [RFC2279] encoding of the ISO 10646 multilingual plane. The charset parameter of the Content-Type header, together with the XML "encoding" attribute, provide charset identification information for MIME and XML processors. Proper use of the charset header with XML is described in RFC 3023. XML also provides a language tagging capability for specifying the language of the contents of a particular XML element. XML uses either IANA registered language tags (see RFC 3066) or ISO 639 language tags in the "xml:lang" attribute of an XML element to identify the language of its content and attributes.

DeltaV applications, since they build upon WebDAV, are subject to the internationalization requirements specified in RFC 2518, Section 16. In brief, these requirements mandate the use of XML character set tagging, character set encoding, and language tagging capabilities. Additionally, they strongly recommend reading RFC 3023 for instruction on the use of MIME media types for XML transport and the use of the charset header.

Within this specification, a label is a human-readable string that is marshaled in the Label header and as XML in request entity bodies. When used in the Label header, the value of the label is URL-escaped and encoded using UTF-8.

16 Security Considerations

All of the security considerations of WebDAV discussed in RFC 2518, Section 17 also apply to WebDAV versioning. Some aspects of the versioning protocol help address security risks introduced by WebDAV, but other aspects can increase these security risks. These issues are detailed below.

16.1 Auditing and Traceability

WebDAV increases the ease with which a remote client can modify resources on a web site, but this also increases the risk of important information being overwritten and lost, either through user error or user maliciousness. The use of WebDAV versioning can help address this problem by guaranteeing that previous information is saved in the form of immutable versions, and therefore is easily available for retrieval or restoration. In addition, the version history provides a log of when changes were made, and by whom. When requests are appropriately authenticated, the history mechanism

provides a clear audit trail for changes to web resources. This can often significantly improve the ability to identify the source of the security problem, and thereby help guard against it in the future.

16.2 Increased Need for Access Control

WebDAV versioning provides a variety of links between related pieces of information. This can increase the risk that authentication or authorization errors allow a client to locate sensitive information. For example, if version history is not appropriately protected by access control, a client can use the version history of a public resource to identify later versions of that resource that the user intended to keep private. This increases the need for reliable authentication and accurate authorization.

A WebDAV versioning client should be designed to handle a mixture of 200 (OK) and 403 (Forbidden) responses on attempts to access the properties and reports that are supported by a resource. For example, a particular user may be authorized to access the content and dead properties of a version-controlled resource, but not be authorized to access the DAV:checked-in, DAV:checked-out, or DAV:version-history properties of that resource.

16.3 Security Through Obscurity

While it is acknowledged that "obscurity" is not an effective means of security, it is often a good technique to keep honest people honest. Within this protocol, version URLs, version history URLs, and working resource URLs are generated by the server and can be properly obfuscated so as not to draw attention to them. For example, a version of "http://foobar.com/reviews/salaries.html" might be assigned a URL such as "http://foobar.com/repo/4934943".

16.4 Denial of Service

The auto-versioning mechanism provided by WebDAV can result in a large number of resources being created on the server, since each update to a resource could potentially result in the creation of a new version resource. This increases the risk of a denial of service attack that exhausts the storage capability of a server. This risk is especially significant because it can be an unintentional result of something like an aggressive auto-save feature provided by an editing client. A server can decrease this risk by using delta storage techniques to minimize the cost of additional versions, and by limiting auto-versioning to a locking client, and thereby decreasing the number of inadvertent version creations.

17 IANA Considerations

This document uses the namespace defined by RFC 2518 for XML elements. All other IANA considerations from RFC 2518 are also applicable to WebDAV Versioning.

18 Intellectual Property

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

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

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

19 Acknowledgements

This protocol is the collaborative product of the authors and the rest of the DeltaV design team: Boris Bokowski, Bruce Cragun (Novell), Jim Dombek (Macromedia), David Durand (INSO), Lisa Dusseault (Xythos), Chuck Fay (FileNet), Yaron Goland, Mark Hale (Interwoven), Henry Harbury (Merant), James Hunt, Jeff McAffer (OTI), Peter Raymond (Merant), Juergen Reuter, Edgar Schwarz (Marconi), Eric Sedlar (Oracle), Bradley Sergeant, Greg Stein, and John Vasta (Rational). We would like to acknowledge the foundation laid for us by the authors of the WebDAV and HTTP protocols upon which this protocol is layered, and the invaluable feedback from the WebDAV and DeltaV working groups.

20 References

- [ISO639] ISO, "Code for the representation of names of languages", ISO 639:1988, 1998.
- [RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", BCP 9, RFC 2026, October 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2277] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.
- [RFC2279] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.
- [RFC2396] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.
- [RFC2518] Goland, Y., Whitehead, E., Faizi, A., Carter, S. and D. Jensen, "HTTP Extensions for Distributed Authoring - WEBDAV", RFC 2518, February 1999.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3023] Murata, M., St. Laurent, S. and D. Kohn, "XML Media Types", RFC 3023, January 2001.
- [RFC3066] Alvestrand, H., "Tags for the Identification of Languages", BCP 47, RFC 3066, January 2001.

Appendix A - Resource Classification

This document introduces several different kinds of versioning resources, such as version-controlled resources, versions, checked-out resources, and version history resources. As clients discover resources on a server, they may find it useful to classify those resources (for example, to make UI decisions on choice of icon and menu options).

Clients should classify a resource by examining the values of the DAV:supported-method-set (see Section 3.1.3) and DAV:supported-live-property-set (see Section 3.1.4) properties of that resource.

The following list shows the supported live properties and methods for each kind of versioning resource. Where an optional feature introduces a new kind of versioning resource, that feature is noted in parentheses following the name of that kind of versioning resource. If a live property or method is optional for a kind of versioning resource, the feature that introduces that live property or method is noted in parentheses following the live property or method name.

A.1 DeltaV-Compliant Unmapped URL (a URL that identifies no resource)

Supported methods:

- PUT [RFC2616]
- MKCOL [RFC2518]
- MKACTIVITY (activity)
- VERSION-CONTROL (workspace)
- MKWORKSPACE (workspace)

A.2 DeltaV-Compliant Resource

Supported live properties:

- DAV:comment
- DAV:creator-displayname
- DAV:supported-method-set
- DAV:supported-live-property-set
- DAV:supported-report-set
- all properties defined in WebDAV [RFC2518].

Supported methods:

- REPORT
- all methods defined in WebDAV [RFC2518]
- all methods defined in HTTP/1.1 [RFC2616].

A.3 DeltaV-Compliant Collection

Supported live properties:

- all DeltaV-compliant resource properties.

Supported methods:

- BASELINE-CONTROL (baseline)
- all DeltaV-compliant resource methods.

A.4 Versionable Resource

Supported live properties:

- DAV:workspace (workspace)
- DAV:version-controlled-configuration (baseline)
- all DeltaV-compliant resource properties.

Supported methods:

- VERSION-CONTROL
- all DeltaV-compliant resource methods.

A.5 Version-Controlled Resource

Supported live properties:

- DAV:auto-version
- DAV:version-history (version-history)
- DAV:workspace (workspace)
- DAV:version-controlled-configuration (baseline)
- all DeltaV-compliant resource properties.

Supported methods:

- VERSION-CONTROL
- MERGE (merge)
- all DeltaV-compliant resource methods.

A.6 Version

Supported live properties:

- DAV:predecessor-set
- DAV:successor-set
- DAV:checkout-set
- DAV:version-name
- DAV:checkout-fork (in-place-checkout or working resource)
- DAV:checkin-fork (in-place-checkout or working resource)
- DAV:version-history (version-history)
- DAV:label-name-set (label)
- DAV:activity-set (activity)
- all DeltaV-compliant resource properties.

Supported methods:

- LABEL (label)
- CHECKOUT (working-resource)
- all DeltaV-compliant resource methods.

A.7 Checked-In Version-Controlled Resource

Supported live properties:

- DAV:checked-in
- all version-controlled resource properties.

Supported methods:

- CHECKOUT (checkout-in-place)
- UPDATE (update)
- all version-controlled resource methods.

A.8 Checked-Out Resource

Supported live properties:

- DAV:checked-out
- DAV:predecessor-set
- DAV:checkout-fork (in-place-checkout or working resource)
- DAV:checkin-fork (in-place-checkout or working resource)
- DAV:merge-set (merge)
- DAV:auto-merge-set (merge)
- DAV:unreserved (activity)
- DAV:activity-set (activity)

Supported methods:

- CHECKIN (checkout-in-place or working-resource)
- all DeltaV-compliant resource methods.

A.9 Checked-Out Version-Controlled Resource (checkout-in-place)

Supported live properties:

- all version-controlled resource properties.
- all checked-out resource properties.

Supported methods:

- UNCHECKOUT
- all version-controlled resource methods.
- all checked-out resource methods.

A.10 Working Resource (working-resource)

Supported live properties:

- all DeltaV-compliant resource properties
- all checked-out resource properties
- DAV:auto-update.

Supported methods:

- all checked-out resource methods.

A.11 Version History (version-history)

Supported live properties:

- DAV:version-set
- DAV:root-version
- all DeltaV-compliant resource properties.

Supported methods:

- all DeltaV-compliant resource methods.

A.12 Workspace (workspace)

Supported live properties:

- DAV:workspace-checkout-set
- DAV:baseline-controlled-collection-set (baseline)
- DAV:current-activity-set (activity)
- all DeltaV-compliant collection properties.

Supported methods:

- all DeltaV-compliant collection methods.

A.13 Activity (activity)

Supported live properties:

- DAV:activity-version-set
- DAV:activity-checkout-set
- DAV:subactivity-set
- DAV:current-workspace-set
- all DeltaV-compliant resource properties.

Supported methods:

- all DeltaV-compliant resource methods.

A.14 Version-Controlled Collection (version-controlled-collection)

Supported live properties:

- DAV:eclipsed-set
- all version-controlled resource properties.

Supported methods:

- all version-controlled resource methods.

A.15 Collection Version (version-controlled-collection)

Supported live properties:

- DAV:version-controlled-binding-set
- all version properties.

Supported methods:

- all version methods.

A.16 Version-Controlled Configuration (baseline)

Supported live properties:

- DAV:baseline-controlled-collection
- all version-controlled resource properties.

Supported methods:

- all version-controlled resource methods.

A.17 Baseline (baseline)

Supported live properties:

- DAV:baseline-collection
- DAV:subbaseline-set
- all version properties.

Supported methods:

- all version methods.

A.18 Checked-Out Version-Controlled Configuration (baseline)

Supported live properties:

- DAV:subbaseline-set
- all version-controlled configuration properties.

Supported methods:

- all version-controlled configuration methods.

Authors' Addresses

Geoffrey Clemm
Rational Software
20 Maguire Road, Lexington, MA 02421

EMail: geoffrey.clemm@rational.com

Jim Amsden
IBM
3039 Cornwallis, Research Triangle Park, NC 27709

EMail: jamsden@us.ibm.com

Tim Ellison
IBM
Hursley Park, Winchester, UK S021 2JN

EMail: tim_ellison@uk.ibm.com

Christopher Kaler
Microsoft
One Microsoft Way, Redmond, WA 98052

EMail: ckaler@microsoft.com

Jim Whitehead
UC Santa Cruz, Dept. of Computer Science
1156 High Street, Santa Cruz, CA 95064

EMail: ejw@cse.ucsc.edu

Full Copyright Statement

Copyright (C) The Internet Society (2002). All Rights Reserved.

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

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

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

Acknowledgement

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

