SUPPORTING ARTICLE

Requirements for Distributed Authoring and Versioning on the World Wide Web

J.A. Slein
Xerox Corporation, Webster, NY

F. Vitali
University of Bologna, Bologna, Italy

E.J. Whitehead, Jr.
U.C. Irvine, Irvine, CA

D.G. Durand
Boston University, Boston, MA

■ Current World Wide Web (WWW or Web) standards provide simple support for applications that allow remote editing of typed data. In practice, the existing capabilities of the WWW have proven inadequate to support efficient, scalable, remote editing, free of overwriting conflicts. A list of features in the form of requirements which, if implemented, would improve the efficiency of common remote editing operations, provide a locking mechanism to prevent overwrite conflicts, improve relationship management support between non-HTML data types, provide a simple attribute-value metadata facility, provide for the creation and reading of container data types, and integrate versioning into the WWW are presented in this article.

Functionality which, if standardized in the context of the WWW, would allow tools for remote loading, editing and saving (publishing) of various media types on the WWW to interoperate with any compliant Web server is described here. As much as possible, this functionality is described without proposing an implementation, since there are many ways to perform the functionality within the WWW framework. It could be implemented in extensions to HTTP, in a new protocol to be layered on top of HTTP, in additional MIME types, or some combination of these and other approaches. It is also possible that a single mechanism could simultaneously satisfy several requirements.

In this article we want to reflect the consensus of the WWW Distributed Authoring and Versioning working group (WebDAV) on the functionality that needs to be standardized to support distributed authoring and versioning on the Web. However, this version still has some problems and questions that are being debated in the working group:

—Should attribute search be in scope?
—Is support for partial-resource locking needed?
—Are other lock types besides write locks needed?
—What are the semantics of locking?
—Semantics of copy and move for resources with attributes or relationships, as well as for collections and version graphs, are necessary.
—What are the implications of attributes, relationships, collections, and versioning for the HTTP DELETE method?
—The semantics for removing resources from collections are problematical.
—Should the goal of simplicity for clients be balanced against allowing versioning servers to implement a variety of versioning policies?
—Are reservations apart from versioning or do they make sense only in versioning systems?

—Are diff and merge needed?
—What requirements are mandatory, and which are optional for a server to implement in order to be WebDAV-compliant?

Rationale

Current Web standards contain functionality that enables editing of Web content from a remote location, without direct access to the storage media via an operating system. This capability is exploited by several existing HTML distributed authoring tools and by a growing number of mainstream applications (e.g., word processors) that allow users to write (publish) their work to an HTTP server. To date, experience with HTML authoring tools has shown they are unable to meet users' needs using the facilities of Web standards. As a consequence, either the introduction of distributed authoring capability is postponed, or nonstandard extensions (developed in isolation they are not interoperable) are added to the HTTP protocol [Fielding et al. 1997] or to other Web standards.

Other authoring applications want to access document repositories or version control systems through Web gateways as well, and have been similarly frustrated. Where access is available at all, it is through nonstandard extensions to HTTP or other standards, forcing clients to use a different interface for each vendor's service.

In this article we describe the requirements for a set of standard extensions to the Web that would allow distributed Web authoring tools to provide the functionality users need, by means of the same standard syntax across all compliant servers. The broad categories of functionality that need to be standardized are: attributes, relationships, locking, reservations, retrieval of unprocessed sources, partial write, name space manipulation, collections, and versioning.

Terminology

Where there is overlap, usage is intended to be consistent with that in the HTTP 1.1 specification [Fielding et al. 1997].

Attribute. Named descriptive information about a resource.

Client. A program that issues HTTP requests and accepts responses.

Collection. A resource that contains other resources, either directly or by reference.

Distributed authoring tool. A program that can retrieve a source entity via HTTP, allow editing of this entity, and then save/publish this entity to a server using HTTP.

Entity. The information transferred in a request or response.

Hierarchical collection. A hierarchical organization of resources. A hierarchical collection is a resource that contains other resources, including collections, either directly or by reference.

Lock. A mechanism for preventing anyone other than the owner of the lock from accessing a resource.

Member of version graph. A resource that is a node in a version graph, and so is derived from the resources that precede it in the graph, and is the basis for those that succeed it.

Relationship. A typed connection between two or more resources.

Reservation. A declaration to the server that one intends to edit a resource.

Resource. A network data object or service that can be identified by a URI.

Server. A program that receives and responds to HTTP requests.

Server attribute. An attribute whose value is generated by the server.

User agent. The client who initiates a request.

User attribute. An attribute whose value is provided by a user or a user agent.

Version graph. A directed acyclic graph with resources as its nodes, where each node is derived from its predecessor(s).

Write lock. A lock that prevents anyone except its owner from modifying the resource it applies to.

General Principles

Here we describe a set of general principles that WebDAV extensions should follow. These principles cut across categories of functionality.

USER AGENT INTEROPERABILITY

All WebDAV clients should be able to work with any WebDAV-compliant HTTP server. It is acceptable for some client/server combinations to provide special features that are not universally available, but the protocol should be sufficient that a basic level of functionality will be universal.

CLIENT SIMPLICITY

The WebDAV extensions should be designed to allow for simple client implementations.

LEGACY CLIENT SUPPORT

It should be possible to implement a WebDAV-compliant server to interoperate with non-WebDAV clients. Such a server would be able to understand any valid HTTP 1.1 request from an ordinary Web client without WebDAV extensions, and to provide a valid HTTP 1.1 response that does not require the client to understand the extensions.

DATA FORMAT COMPATIBILITY

WebDAV-compliant servers should be able to work with existing resources and URIs [Berners-Lee et al. 1994]. (Special information should not become a mandatory part of a document.)

REPLICATED, DISTRIBUTED SYSTEMS

Distribution and replication are at the heart of the Internet. All WebDAV extensions should be designed to allow for distribution and replication. Version trees should be able to be split across multiple servers. Collections may have members on different servers. Resources may have attributes on different servers. Any resource may be cached or replicated for mobile computing or for other reasons. Consequently, the WebDAV extensions must be able to operate in a distributed, replicated environment.

PARSIMONY IN CLIENT-SERVER INTERACTIONS

The WebDAV extensions should keep to a minimum the number of interactions between the client and the server needed to perform common functions. For example, publishing a document to the Web will often mean publishing content together with related metadata. A client may often need to find out what version graph a particular resource belongs to, or to find out which resource in a version graph is the published one. The extensions should make it possible to do this efficiently.

ALTERNATE TRANSPORT MECHANISMS

It may be desirable to transport WebDAV requests and responses by other mechanisms, particularly email, in addition to HTTP. The design of the WebDAV extensions should take alternative transports into account.

Requirements

In the requirement descriptions the requirements are stated, followed by their rationales.

ATTRIBUTES

Functional Requirements. It must be possible to create, modify, query, read and delete arbitrary attributes on resources of any media type.

Attributes are resources that may have attributes of their own and may be subject to content negotiation, and so forth.

Attributes have implications for the semantics of move, copy, and delete operations. (See “Name Space Manipulation” below.)

Rationale. Attributes describe resources of any media type. They may include bibliographic information such as author, title, publisher, and subject, constraints on usage, PICS ratings, and so on. These attributes have many uses, e.g., supporting searches on attribute values, enforcing copyrights, and creating catalog entries as placeholders for objects not available in electronic form, or that will be available later.

RELATIONSHIPS

Functional Requirements. It must be possible to create, modify, query, read and delete typed relationships between resources of any media type.

Relationships have implications for the semantics of move, copy, and delete operations. (See “Name Space Manipulation” below.)

Rationale. The hypertext link is one type of relationship between resources. It is browsable using a hypertext style point-and-click user interface. Relationships, whether browsable hypertext links, or simply a means of capturing a connection between resources, have many purposes. Relationships can support push-button printing of a multiresource document in a prescribed order, jumping to the access control page for a resource, and quick browsing of related information (a table of contents, an index, a glossary, help pages, etc.). While relationship support is provided by the HTML “LINK” element, it is limited to HTML resources only [Berners-Lee and Connolly 1995]. Similar support is needed for bitmap image types and other non-HTML media types.

LOCKING

General Principles

Independence of locks. It must be possible to lock a resource without rereading the resource and without committing to editing the resource.

Multiresource locking. It must be possible to take out a lock on multiple resources in the same action, and this locking operation must be atomic across resources.

Partial-resource locking. It must be possible to take out a lock on a subsection of a resource.

Optional server support for locking. Some systems use other mechanisms besides locking to ensure consistency in environments where several users may wish to edit a resource at the same time. These strategies must be permitted.

Functional Requirements

Write locks. It must be possible to restrict modification of a resource to a specific person.

Lock query. It must be possible to find out whether a given resource has any active modification restrictions, and, if so, who currently has modification permission.

Unlock. It must be possible to remove a lock. Only the owner of a lock or a principal with appropriate access rights may remove the lock.

Rationale. At present the Web provides limited support for preventing two or more people from overwriting each other’s modifications when they save to a given URI. Furthermore, there is no way to discover whether someone else is currently making modifications to a resource. This is known as the “lost update problem,” or the “overwrite problem.” Since there can be significant cost associated with discovering and repairing lost modifications, preventing this problem is crucial for supporting distributed authoring. A write lock ensures that only one person may modify a resource, preventing overwrites. Furthermore, locking support is a key component of many versioning schemes, a desirable capability for distributed authoring.

An author may wish to lock an entire web of resources, even though editing just a single resource, to keep the other resources from changing. In this way an author can ensure that if a local hypertext web is consistent in his distributed authoring tool, it will then be consistent when the author writes it to the server. So it should be possible to take out a lock without causing transmission of the resource's contents.

It is often necessary to guarantee that a lock or unlock operation occurs at the same time across multiple resources, a feature supported by the multiple-resource locking requirement. This is useful for preventing a collision between two people trying to establish locks on the same set of resources, since with multiresource locking, one of the two people will get a lock. If this same multiple-resource locking scenario was repeated by using atomic lock operations iterated across the resources, the result would be a splitting of the locks between the two people, based on resource ordering and race conditions.

Partial resource locking provides support for collaborative editing applications, where multiple users may be editing the same resource simultaneously. Partial resource locking also allows multiple people to simultaneously work on a database type resource.

RESERVATIONS

Functional Requirements

Reserve. It must be possible to notify the server that a resource is about to be edited by a given person.

Reservation query. It must be possible to find out whether a given resource has any active reservations, and if so, who currently holds reservations.

Release reservation. It must be possible to release the reservation. Only the owner of a reservation or a principal with appropriate access rights may release the reservation.

Rationale. Experience in configuration management systems has shown that people need to know when they are about to enter a parallel editing situation. Once notified, they either decide not to edit in parallel with other authors, or they use out-of-band communication (face-to-face, telephone, etc.) to coordinate their editing to minimize the difficulty of merging their results. Reservations are separate from locking, since a write lock does not necessarily imply a resource will be edited, and a reservation does not carry with it any access restrictions. This capability supports versioning, since a check-out typically involves taking out a write lock, making a reservation, and getting the resource to be edited.

RETRIEVAL OF UNPROCESSED SOURCE FOR EDITING

Functional Requirement. The source of any given resource must be retrievable.

Rationale. There are many cases where the source stored on a server does not correspond to the actual entity transmitted in response to an HTTP GET. Current known cases are server side, including directives and Standard Generalized Markup Language (SGML) source that is converted on the fly to Hypertext Markup Language (HTML) output. There are many possible cases, such as automatic conversion of bitmap images into several variant bitmap media types (e.g., GIF, JPEG) and automatic conversion of an application's native media type into HTML. As an example of this last case, a word processor could store its native media type on a server that automatically converts it to HTML. A GET of this resource would retrieve the HTML. Retrieving the source would retrieve the word processor native format.

This requirement should be met by a general mechanism that can handle both the "single-step" source processing described above, where the source is converted into the transmission entity via a single conversion step, as well as "multistep" source processing, where there is one or more intermediate processing steps and outputs. An example of multistep source processing is the relationship between an executable binary image, its object files, and its source language files. Note that the relationship between the source and transmission entity could be expressed using the relationship functionality described in "Relationships", page 19.

PARTIAL WRITE

Functional Requirement. After editing a resource, it must be possible to write only the changes to the resource, rather than retransmitting the entire resource.

Rationale. During distributed editing which occurs over wide geographic areas and/or over low bandwidth connections, it is extremely inefficient and frustrating to rewrite a large resource after minor changes, such as a one-character spelling correction. Support is needed for transmitting "insert" (e.g., add this sentence in the middle of a document) and "delete" (e.g., remove this paragraph from the middle of a document) style updates. Support for partial resource updates will make small edits more efficient and allow distributed authoring tools to scale up for editing large documents.

NAME SPACE MANIPULATION

Copy

Functional Requirements. It must be possible to duplicate a resource without a client first loading and then resaving the resource. After the copy operation, the content of the destination resource must be octet-for-octet identical to the content of the source resource. A modification to either resource must not cause a modification to the other. The copy operation should leave an audit trail.

It must be possible for a client to specify whether a resource's user attributes and relationships are to be copied with it, although the server may decline to copy them. It may decline to copy user attributes if the destination namespace supports different attributes than the source namespace does, for example. The server may follow whatever policy it likes for copying server attributes.

Copying a collection causes all of the resources that belong to it directly to be copied as well. For resources that belong to it by reference, the reference is copied. It must be possible for a client to specify whether subcollections should be copied with the collection.

If a version graph is copied, all relationships between nodes in the graph must be changed in the new copy to reflect its new location.

Rationale. There are many reasons why a resource might need to be duplicated, such as changing ownership, preparing for major modifications, or making a backup. Due to network costs associated with loading and saving a resource, it is far preferable to have a server perform a resource copy than have a client do so. If a copied resource records which resource it is a copy of, then it would be possible for a cache to avoid loading the copied resource if it already locally stores the original.

Move/Rename

Functional Requirements. It must be possible to change the location of a resource without a client loading and then resaving the resource under a different name. After the move operation, the content of the resource at its new location must be octet-for-octet identical to the content of the prior resource. It must no longer be possible to access the resource at its original location. The move operation should leave an audit trail.

It must be possible for a client to specify whether a resource's user attributes and relationships are to be moved with it, although the server may decline to move them. It may decline to move user attributes if the destination namespace supports different attributes than the source namespace, for example. The server may follow whatever policy it likes for server attributes.

Moving a collection causes all of the resources that belong to it directly to be moved as well. For resources that belong to it by reference, the reference is moved also. It must be possible for a client to specify whether subcollections should be moved with the collection. If not, subcollections that belong to the collection directly should be deleted from the source location.

If a version graph is moved, all relationships between nodes in the graph must be changed in the destination resource to reflect its new location.

Rationale. It is often necessary to change the name of a resource, for example due to adoption of a new naming convention, or if a typing error was made in entering the name originally. Due to network costs, it is undesirable to perform this operation by loading and then resaving the resource, followed by a delete of the old resource. Similarly, a single rename operation is more efficient than a copy followed by a delete operation. Note that moving a resource is considered the same function as renaming a resource. The audit trail makes it possible for the server to redirect client requests for the resource at its old location, perhaps with a "301 Moved Permanently" status code.

Delete

HTTP already provides a DELETE method, but the semantics of DELETE must be reconsidered once attributes, relations, collections, and versions are introduced.

When a resource is deleted, it must be possible for a client to specify whether its attributes are to be deleted with it. In an environment where resources may share the same attributes, the server may decline to delete the attributes.

When a resource is deleted, the relationships in which it participates should also be deleted.

If the resource being deleted is a collection, all resources that belong to it directly will be deleted as well. Resources that belong to it by reference remain unaffected.

If the resource being deleted is a member of a version graph, the predecessor and successor relationships in the graph must be updated, and any metadata required by the versioning server must be supplied. The versioning server may, for example, require a comment explaining the reason for the deletion.

COLLECTIONS

A collection is a resource that is a container for other resources, including other collections. A resource may belong to a collection either directly or by reference. If a resource belongs to a collection directly, namespace operations like copy, move, and delete applied to the collection also apply to the resource. If a resource belongs to a collection by reference, namespace operations applied to the collection affect only the reference, not the resource itself.

Functional Requirements

List collection. A listing of all resources in a specific collection must be accessible.

Make collection. It must be possible to create a new collection.

Add to collection. It must be possible to add a resource to a collection directly or by reference.

Remove from collection. It must be possible to remove a resource from a collection. In the case of a resource that belongs to the collection directly, this results in the resource being deleted. In the case of a resource that is merely referenced by the collection, only the reference is removed.

Collections have implications for the semantics of move, copy, and delete operations. (See "Name Space Manipulation," p. 20).

Rationale. Berners-Lee et al. [1994] state that "some URL schemes (such as the ftp, http, and file schemes) contain names that can be considered hierarchical." Especially for HTTP servers, which directly map all or part of their URL name space into a filesystem, it is very useful to get a listing of all resources located at a particular hierarchy level. This functionality supports "Save As . . ." dialog boxes, which provide a listing of the entities at a current hierarchy level and allow navigation through the hierarchy. It also supports the creation of graphical visualizations (typically as a network) of the hypertext structure among the entities at a hierarchy level, or set of levels. It also supports a tree visualization of the entities and their hierarchy levels.

In addition, document management systems may want to make their documents accessible through the Web. They typically allow the organization of documents into collections, and so also want their users to be able to view the collection hierarchy through the Web.

There are many instances where there is not a strong correlation between a URL hierarchy level and the notion of a collection. One example is a server in which the URL hierarchy level maps to a computational process that performs some resolution on the name. In this case the contents of the URL hierarchy level can vary, depending on the input to the computation—and the number of resources accessible via the computation can be very large. It does not make sense to implement a directory feature for such a namespace. However, the utility of listing the contents of those URL hierarchy levels that do correspond to collections, such as the large number of HTTP servers that map their namespace to a filesystem, argue for the inclusion of this capability; although it will not be meaningful in all cases. If listing the contents of a URL hierarchy level does not make sense for a particular URL, then a "405 Method Not Allowed" status code could be issued.

The ability to create collections to hold related resources supports management of a name space by packaging its members into small, related clusters. The utility of this procedure is demonstrated by the broad implementation of directories in recent operating systems. The ability to create a collection also supports the creation of "Save As . . ." dialog boxes with "New Level/Folder/Directory" capability, common in many applications.

VERSIONING

Background and General Principles

Stability of versions. Most versioning systems are intended to provide an accurate record of the history of the evolution of a document. Accuracy is ensured by the fact that a version eventually becomes "frozen" and immutable. Once a version is frozen, further changes will create new versions rather than modifying the original. In order for caching and persistent references to be properly maintained, a client must be able to determine that a version has been frozen. Any successful attempt to retrieve a frozen version of a resource will always retrieve exactly the same content, or return an error if that version (or the resource itself) is no longer available.

Operations for creating new versions. Version management systems vary greatly in the operations they require, the order of the operations, and how they are combined into atomic functions. In the most complete cases, the logical operations involved are:

—reserve existing version
—lock existing version
—retrieve existing version
—request or suggest identifier for new version
—write new version
—release lock
—release reservation

With the exception of requesting a new version identifier, all of these operations have applications outside of versioning and are either already part of HTTP or are discussed in earlier sections of these requirements. Typically, versioning systems combine reservation, locking, and retrieval, or some subset of these, into an atomic checkout function. They combine writing, releasing the lock, and releasing the reservation, or some subset of these, into an atomic check-in function. The new version identifier may be assigned either at check-out or at check-in.

The WebDAV extensions must find some balance between allowing versioning servers to adopt whatever policies they wish with regard to these operations and enforcing enough uniformity to keep client implementations simple.

The versioning model. Each version typically stands in a "derived from" relationship to its predecessor(s). It is possible to derive several different versions from a single version (branching), and to derive a single version from several versions (merging). Consequently, the collection of related versions forms a directed acyclic graph. In the following discussion, this graph will be called a "version graph." Each node of this graph is a "version" or "member of the version graph." The arcs of the graph capture the "derived from" relationships.

It is also possible for a single resource to participate in multiple version graphs.

The WebDAV extensions must support this versioning model—though particular servers may restrict it in various ways.

Versioning policies. Many writers, including Feiler [1991] and Haake and Hicks [1996], have discussed the notion of versioning styles (referred to here as versioning policies, to reflect the nature of client/server interaction) as one way to think about the different policies that versioning systems implement. Versioning policies include decisions on the shape of version histories (linear or branched), the granularity of change tracking, locking requirements made by a server, and so on. The protocol should clearly identify those policies that it dictates and those policies left up to versioning system implementors or administrators.

It is possible to version resources of any media type.

Functional Requirements

Referring to a version graph. There must be a way to refer to a version graph as a whole.

Some queries and operations apply not just to one member of a version graph, but to the version graph as a whole. For example, a client may request that an entire graph be moved, or may ask for a version history. In these cases a way to refer to the whole version graph is required.

Referring to a specific member of a version graph. There must be a way to refer to each member of a version graph. This means that each member of the graph is itself a resource.

Each member of a version graph must be a resource if it is to be possible for a hypertext link to refer to specific version of a page, or for a client to request a specific version of a document for editing.

A client must be able to determine whether a resource is a version graph, or whether a resource is itself a member of a version graph.

A resource may be a simple, nonversioned resource, or it may be a version graph, or it may be a member of a version graph. A client needs to be able to tell which sort of resource it is accessing.

There must be a way to refer to a server-defined default member of a version graph.

The server should return a default version of a resource for requests that ask for the default version, as well as for requests where no specific version information is provided. This is one of the simplest ways to guarantee nonversioning client compatibility. This does not rule out the possibility of a server returning an error when no sensible default exists.

It may also be desirable to be able to refer to other special members of a version graph. For example, there may be a current version for editing that is different from the default version. For a graph with several branches, it may be useful to be able to request the tip version of any branch.

It must be possible, given a reference to a member of a version graph, to find out which version graph(s) that resource belongs to.

This makes it possible to understand the versioning context of the resource, to retrieve a version history for the graphs to which it belongs, and to browse the version graph. It also supports some comparison operations—making it possible to determine whether two references designate members of the same version graph.

Navigation of a version graph. Given a reference to a member of a version graph, it must be possible to discover and access the following related members of the version graph.

—root member of the graph
—predecessor member(s)
—successor member(s)
—default member of the graph

It must be possible in some way for a versioning client to access versions related to a resource currently being examined.

Version topology. There must be a way to retrieve the complete version topology for a version graph, including information about all members of the version graph. The format for this information must be standardized, so that the basic information can be used by all clients. Other specialized formats should be accommodated for servers and clients that require information that cannot be included in the standard topology.

A client must be able to request that the server generate a version identifier for a new member of a version graph. Such an identifier will not be used by any other client in the meantime. The server may refuse the request.

A client must be able to propose a version identifier to be used for a new member of a version graph. The server may refuse to use the client's suggested version identifier.

A version identifier must be unique across a version graph.

A client must be able to supply version-specific metadata to be associated with a new member of a version graph. (See "Attributes," p. 19) At a minimum, it must be possible to associate comments with the new member, explaining what changes were made.

A client must be able to query the server for information about a version tree, including which versions are locked, which are reserved for editing, and by whom (Session Tracking).

It must be possible for a client to get from the server a list of the differences between two or more resources of the same media type.

A client must be able to request that the server merge two or more resources, and return the result of the merge to the client or store the result as a resource. Server support for this functionality is optional.

Versioning has implications for the semantics of move, copy, and delete operations. (See "Name Space Manipulation," p. 20.) In addition, if the WebDAV extensions allow versioning servers to PUT or POST new members into a version graph, the semantics of those methods must be extended to encompass the new functionality.

Rationale. Versioning in the context of the world wide web offers a variety of benefits:

It provides infrastructure for efficient and controlled management of large evolving Web sites. Modern configuration management systems are built on some form of repository that can track the revision history of individual resources, and provide the (higher-level) tools to manage those saved versions. Basic versioning capabilities are required to support such systems.

It allows parallel development and update of single resources. Since versioning systems register change by creating new objects, they enable simultaneous write access by allowing the creation of variant versions. Many also provide merge support to ease the reverse operation.

It provides a framework for coordinating changes to resources. While specifics vary, most systems provide some method of controlling or tracking access to enable collaborative resource development.

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.

It provides 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.

It allows explicit semantic representation of single resources with multiple states. A versioning system directly represents the fact that a resource has an explicit history, and a persistent identity across the various states it has had during the course of that history.

AUTHENTICATION AND SECURITY

The WebDAV extensions should make use of existing authentication and security protocols. The (WebDAV) specification must state how the WebDAV extensions interoperate with existing authentication and security schemes. **SV**

Acknowledgements

Our understanding of these issues has emerged as the result of much thoughtful discussion, email, and assistance by many people, who deserve recognition for their effort: Martin Cagan, Continuus Software <Marty_Cagan@continuus.com>; Steve Carter, Novell <srcarter@novell.com>; Dan Connolly, World Wide Web Consortium <connolly@w3.org>; Jim Cunningham, Netscape <jfc@netscape.com>; Mark Day, Lotus <Mark_Day@lotus.com>; Asad Faizi, Netscape <asad@netscape.com>; Ron Fein, Microsoft <ronfe@microsoft.com>; David Fiander, Mortice Kern Systems <davidf@mks.com>; Roy Fielding, U.C. Irvine <fielding@ics.uci.edu>; Yaron Y. Goland, Microsoft <yarong@microsoft.com>; Phill Hallam-Baker, MIT <hallam@ai.mit.edu>; Dennis Hamilton, Xerox PARC <hamilton@parc.xerox.com>; Andre van der Hoek, University of Colorado, Boulder <andre@bigtime.cs.colorado.edu>; Del Jensen, Novell <dcjensen@novell.com>; Gail Kaiser, Columbia University <kaiser@cs.columbia.edu>; Rohit Khare, World Wide Web Consortium <khare@w3.org>; Dave Long, America Online <dave@sb.aol.com>; Henrik Frystyk Nielsen, World Wide Web Consortium <frystyk@w3.org>; Ora Lassila, Nokia Research Center <ora.lassila@research.nokia.com>; Larry Masinter, Xerox PARC <masinter@parc.xerox.com>; Murray Maloney, SoftQuad <murray@sq.com>; Jim Miller, World Wide Web Consortium <jmiller@w3.org>; Keith Moore, University of Tennessee, Knoxville <moore@cs.utk.edu>; Andrew Schulert, Microsoft <andyschu@microsoft.com>; Christopher Seiwald, Perforce Software <seiwald@perforce.com>; Einar Stefferud <stef@nma.com>; Richard Taylor, U.C. Irvine <taylor@ics.uci.edu>; and Robert Thau, MIT <rst@ai.mit.edu>.

References

BERNERS-LEE, T., AND CONNOLLY, D. 1995. HyperText markup language specification, 2.0. RFC 1866, MIT/LCS, Nov. 1995

BERNERS-LEE, T., MASINTER, L., AND MCCAHILL, M. 1994. Uniform Resource Locators (URL). RFC 1738, CERN, Xerox PARC, Univ. of Minnesota, Dec. 1994.

FEILER, P. 1991. Configuration management models in commercial environments. Software Engineering Inst. Tech. Rep. CMU/SEI-91-TR-7. <http://www.sei.cmu.edu/products/publications/91.reports/91.tr.007.html>.

FIELDING, R., GETTYS, J., MOGUL, J.C., FRYSTYK, H., AND BERNERS-LEE, T. 1997. Hypertext transfer protocol—HTTP/1.1. RFC 2068, U.C. Irvine, DEC, MIT/LCS, Jan. 1997.

HAAKE, A., AND HICKS, D. 1996. VerSE: Towards Hypertext versioning styles. In Proceedings of Hypertext '96, The Seventh ACM Conference on Hypertext, ACM, New York, 1996, 224–234.

Status. This document is an Internet draft, which are working documents of the Internet Engineering Task Force (IETF), its areas and its working groups. Note that other groups may also distribute working information as Internet drafts.

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

To learn the current status of any Internet draft, check the "lid-abstracts.txt" listing contained in the Internet drafts shadow directories on ftp.is.co.za (Africa), nic.nordu.net (Europe), munnari.oz.au (Pacific Rim), ds.internic.net (U.S. East coast) or ftp.isi.edu (U.S. West coast). Further information about the IETF can be found at URL: http://www.ietf.org/

Distribution of this document is unlimited. Please send comments to the WWW Distributed Authoring and Versioning (WebDAV) mailing list, <w3c-dist-auth@w3.org>, which may be joined by sending a message with subject "subscribe" to <w3c-dist-auth-request@w3.org>. Discussions are archived at URL: http://www.w3.org/pub/WWW/Archives/Public/w3c-dist-auth/.