This document specifies the general design of the Versioning WebDAV Apache server module. Additionally it specifies the interfaces between the components as well as identifies the risks incurred by the design decisions. Furthermore it identifies the feature package implemented by the server.

1 Feature Set

The DeltaV draft specifies several feature packages which can be implemented separate of each other. Our long-term goal is to implement the Basic-Server-Workspace package. This allows parallel development of version-controlled resources, where the server maintains all the persistent information, as opposed to the client as in the Basic-Client-Workspace package. Our choice of the initial package does not hinder the future implementation of the Basic-Client-Workspace as well as advanced versioning features, since all can exist in parallel. The decision of the feature set is based on the fact that it facilitates the development of clients, thus easing the acceptance, since the server removes the overhead as well as complexity of maintaining all the persistent information.

However, our initial release will contain only auto-versioning functionality which is sufficient to allow non-versioning WebDAV clients to make use of versioning. Thus, we will not spend time on unusable features before we have a product in the marketplace.

2 Components

The versioning server is partitioned into two components, namely the Apache and Storage Backend components. Each of the aforementioned components deals with a distinct aspect of the execution and validation of a versioning request.

2.1 Apache Component

Due to the strong ties between DeltaV and WebDAV the versioning functionality will be incorporated into the mod_dav Apache module. Since at the time of the development of the mod_dav module the DeltaV specification was still in it’s early stages, the module merely contains hooks for the versioning functions. Thus the development of this component, given the existent hook functions and infrastructure, will entail the validation of the versioning HTTP requests. Validation in terms of ensuring the request adheres to the syntax of the versioning RFC as well as satisfies the pre-conditions for each of the methods e.g. that a resource specified in the URI of a VERSION-CONTROL request identifies a versionable resource.
2.2 Storage Backend Component

The main task of the storage backend component is to allocate the necessary files, such as the versions, version histories, as well as maintain their associated properties. Until recently, we believed that reusing CVS would simplify implementation by handling much of the administrative overhead of maintaining versions. However, the versioning standard requires that this metadata be stored in the XML property database associated with each file. Thus, CVS becomes useful mainly as a more compact, but slower storage system. Further, implementing the backend via CVS would incur a significantly higher amount of development effort and time, as well as require a detailed understanding of the general Apache and mod_dav module architecture, both of which contain only limited documentation.

Due to time constraints, as well as limited development resources, we have therefore decided to implement only a simpler (and faster) flat file system storage component. Implementing the storage backend in this fashion does not limit the usefulness of the versioning server. It simply entails a slightly larger use of storage space since each version will exist in its entirety, as opposed to merely the diffs for CVS.

3 Execution Order

The execution order for a given request is fairly simple. Assume a request is made for a versioning method e.g. CHECKOUT. Apache will query all of the available modules in order to determine if one is available to handle this request. Thus mod_dav receives the request, and it’s related information, and gains control. Based on the request method mod_dav dispatches the appropriate request handler from within it’s module. In the example it would call dav_method_checkout. The function called then validates the request and in the case of a versioning request calls the appropriate storage backend function.

4 Interfaces

4.1 Apache → Versioning mod_dav Functions

The mod_dav module defines a clear interface between it and each of the method handlers via a dispatch table which calls the appropriate method handlers of the form static int dav_method_X(request_rec *r), where X is any of the methods defined in the DeltaV and WebDAV specifications e.g. static int dav_method_versioncontrol(request_rec *r) is the function used to handle the version control requests.

4.2 Versioning Functions → Storage Backend

The interface between the module and the storage backend is defined in a per versioning method function which gets passed the request in it’s entirety. All of the interfaces will be of the form: dav_error *(<METHOD NAME>)(dav_resource *resource);

5 Parallel Development

Due to the two component nature of the server parallel development should prove fairly easy. More precisely since there are two development members in the team, development can be partitioned into a Apache Component engineer and a Storage Backend engineer. Since the two components will only interact using a well defined interface, they can be developed in parallel.
6 Risks

6.1 Feature Set Selection

The selection of the initial feature set could impede the acceptance of the software. More precisely while the selection of the Basic-Server-Workspace eases the development of the clients, since all the persistent information is stored on the server, this could dissuade administrators from installing the software due to the increased storage requirements. But given that storage is getting increasingly cheaper, we believe this to be a non-issue.

6.2 Testability

Without any known versioning clients, we can only test the system with hand-crafted test cases. This means that the tests are likely to share the same assumptions that affect our implementation. Since later client implementations by others may use different interpretations of the standard, it is important that we document and attempt to resolve ambiguities in the standard. A significant portion of our time has already been spent on doing just that.