

-continued

- 
- 11 Name: nonversioned  
 Namespace: DAV:  
 Purpose: Used to indicate that an update to any property that is not version-specific in the containing object should cause the property's contents to be set to the empty string.  
 Description: The nonversioned XML element indicates that an update to any property that is not version-specific should cause the version-specific property's contents to be set to the empty string. This includes both mutable and immutable properties.  
 <!ELEMENT nonversioned EMPTY>
- 12 Name: nonreplicator  
 Namespace: DAV:  
 Purpose: Used to indicate that an update to any property that is not a replicator class, version-specific property in the containing object should cause the property's contents to be set to the empty string.  
 Description: The nonreplicator XML element indicates that an update to any property that is not a replicator class, version-specific property should cause the version-specific property's contents to be set to the empty string. This includes both mutable and immutable properties, as well as some version-specific properties. A replicator class, version-specific property would likely choose this as its inclusionlist setting.  
 <!ELEMENT nonreplicator EMPTY>
- 13 Name: exclusionlist  
 Namespace: DAV:  
 Purpose: Specifies a list of properties for which an update should not cause the property's contents should be set to the empty string.  
 Description: The exclusionlist XML element specifies the list of properties for which an update should not cause the version-specific property's contents to be set to the empty string. The list can include the special keywords immutable, nonversioned, and nonreplicator as well as a list of URIs of named properties. The version-specific property itself is always considered to be a part of its own exclusion list. That is, update to this property will never cause it to be invalidated. The exclusion list, if any, has precedence over the inclusion list.  
 <!ELEMENT exclusionlist (updatetype, (immutable | nonversioned | nonreplicator href+))>
- 

[0052] In order to actually create a version specific property, the third party application uses standard DAV mechanisms, such as PROPPATCH, PUT, etc. These mechanisms are described in more detail in the Appendix A. In alternative embodiments, these version specific properties are created using other methods.

[0053] Although the collection 300 is shown and described as having only one version specific property, the collection 300 may have other properties, including other version specific properties as well. Indeed, several third party applications may request that persistent state information be in association with and thus numerous version specific properties, such as property 306 may be created and stored along with collection 300.

[0054] The exemplary physical environment having now been discussed, the remaining part of this description section will be devoted to a description of the operative modules embodying aspects of the invention. The logical operations of the various embodiments of the present invention are implemented (1) as a sequence of computer implemented steps or program modules running on a computing system and/or (2) as interconnected hardware or logic modules within the computing system. The implementation is a matter of choice dependent on the performance requirements of the computing system implementing the invention. Accordingly, the logical operations making up the embodiments of the present invention described herein are referred to alternatively as operations, steps or modules.

[0055] FIG. 4 is a flow chart of the operational characteristics related to accessing an object according to aspects of the present invention. Prior to the beginning of flow 400 an object such as objects 204, 206, 208 or 210 shown in

FIG. 2 exists within an XML store, such as store 202 (FIG. 2). In an embodiment of the invention, once an object has been created, then any attempt to access that object initiates the flow 400 shown and described with respect to FIG. 4. Indeed, process 400 begins with access attempt 402, wherein the access attempt 402 relates to any read, execution, or update to an object. The access attempt may be performed by the third party application, such as application 224 (FIG. 2) or by the services layer 236 (FIG. 2).

[0056] Following access attempt 402, determination act 404 determines whether the access is an invalidating access. Determining whether or not an access is an invalidating access involves an evaluation of mask information within each version-specific property, assuming there is more than one, of the particular object being accessed. Evaluating the mask information provides what types of accesses to that object require that the particular version-specific property should be invalidated. Consequently, comparing the mask information to the actual access attempt provides whether the version-specific property should be invalidated. When such a relevant-update access attempt is identified by determination act 404, flow branches YES to invalidate operation 406.

[0057] In an embodiment of the invention the version-specific property is created and used by a virus-scan application. In such a situation, determination act 404 determines whether the access attempt is a relevant update access based on a criteria set by the virus-scan application. That is, the virus-scan application has predetermined what a relevant update access is, such as whether the access attempt will modify actual object data by changing old data or adding new data. For example, a particular virus-scan application may want its version-specific property to be invalidated