Containment/Membership triples management

Membership triples management

1. Membership triples shall not be persisted to stored RDF files
2. Simple search index will be populated with membership triples
3. Resource Index shall track which predicates are membership predicates per resource.
    a. This is determined by locating all Indirect and Direct containers with inbound references to the resource, and recording the values of ldp:hasMemberRelation, ldp:isMemberOfRelation and ldp:insertedContentRelation.
4. When performing a GET operation, if the prefer headers indicate that membership relations should be returned, then they will be pulled from the simple search index based on the recorded membership predicates from the index for that resource.
    a. These properties would be merged into the resultant RDF stream prior to returning it to the client.
5. When updating a resource, any properties using a predicate defined as a membership predicate for that resource would be ignored during persistence.
    a. this would prevent clients from providing predicates with the same predicate as a membership predicate. *

* This could result in data loss if clients are attempting to provide the same membership predicates to the same resource in two separate ways. The alternative would be to pull all existing membership references and determine if any of the incoming membership relations are new. It is doable, but potentially costly.

Containment triples management

1. Containment triples (ldp:contains) would not be persisted to stored RDF files.
2. Child resources would store and persist a reference to their containing parent.
    a. this could be fedora:hasParent, or some other predicate. LDP does not define this.
3. Simple search index will be populated with ldp:contains triples
4. When performing a GET operation, if prefer headers indicate that containment relations should be returned, then they will be pulled from the simple search index.
    a. These properties would be merged into the resultant RDF stream prior to returning to the client.

The intention of this approach is:

- to avoid triggering changes to the persisted state of a resource and its RDF when containment/membership changes.
- to avoid having extremely large persisted RDF files, which could impact system performance during read and write operations.
- to avoid unnecessary retrieval of containment/membership properties when they are not requested, as this would greatly reduce the value of the "Prefer: minimal" header.