Audit Service Implementation Proposal

The proposed implementation of the audit service is to use the existing eventing system, Camel workflow engine, and external triplestore.

Phase 1

The first phase of implementation will be to use the existing event system to emit messages about audit events, process those events with Camel, and creating RDF for events in an external triplestore. The primary goal of this phase is to satisfy the audit service requirements with minimal impact on the repository.

1. Make sure that all internal audit events generate JMS messages
2. Make sure that generated messages contain enough information to create event RDF
3. Create Camel workflow to process messages and create event RDF in an external triplestore
4. Propose RDF classes and properties that event RDF should use
5. Document recipe for creating event RDF for external events in an external triplestore using SPARQL Update
6. Document recipe for disabling deleting event triples from external triplestore
7. Document end-to-end recipe for configuring event service
8. Verify that all audit service requirements are satisfied

Phase 2

The second phase of the implementation will be to create an optional component for persisting audit information in the repository. The primary goal of this phase is to improve the durability of the audit persistence using the repository.

1. Create a REST API endpoint for audit events attached to each resource, which allows creating external events and retrieving all events
2. Update the repository to create audit event records in this container for internal events
3. Create configurable option to allow or disallow deleting events in the repository
4. Make sure that other repository functionality is not impacted by enabling or disabling in-repository audit event persistence
5. Document end-to-end recipe for configuring event service with in-repository audit event persistence

Phase 2 Revised

Instead of creating a new REST API, we could simply create a container named, e.g., "audit" within any container. External events could be posted there using the existing LDP API, and we would need to update Fedora 4 to do that automatically for internal events. Much of the machinery needed to do this is already in place as part of the JMS module which currently listens to JCR events emits JMS events for all repository updates. We could either update the JMS module to also create audit nodes, or create a separate module just for listening to JCR events and creating audit nodes.

- A separate module has the advantage of being completely decoupled from the JMS module, which is particularly desirable for an optional module.
- Updating the JMS module has the advantage of being a smaller update to existing code, and making it easier to suppress JMS events related to creating audit nodes.

RDF Vocabulary

Following the Audit Service PROV-O vs PREMIS guidelines, a typical event encoded in RDF would look like this:

```
@prefix audit:  <http://fedora.info/definitions/v4/audit#> .
@prefix fedora: <http://fedora.info/definitions/v4/repository#> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
@prefix premis: <http://www.loc.gov/premis/rdf/v1#> .
@prefix prov:   <http://www.w3.org/ns/prov#> .
@prefix xsd:    <http://www.w3.org/2001/XMLSchema#> .

<event1> a prov:InstantaneousEvent, premis:Event, audit:InternalEvent ;
  premis:hasEventRelatedAgent "jquser"^^xsd:string, "Client Software v1.2.3"^^xsd:string .
  premis:hasEventType <http://id.loc.gov/vocabulary/preservationEvents/cre> ;
  premis:hasEventRelatedObject <http://localhost:8080/rest/55/59/ec/05/5559ec05-6ab1-4d61-905a-a5f3da360b23> ;
  premis:hasEventDateTime "2012-04-30T20:40:40"^^xsd:dateTime .
```

External events (either directly added to a triplestore or created using the REST API in phase 2), should include the rdf:type audit:ExternalEvent to differentiate them from internal events.

Fixity events will also include the checksum generated:

```
<event1> premis:hasFixity <event1#fixity1> ;
  premis:EventOutcomeInformation "SUCCESS" .

<event1#fixity1> a premis:Fixity ;
  premis:hasMessageDigest "cf23df2207d99a74fbe169e3eba035e633b65d94"^^xsd:string ;
  premis:hasMessageDigestAlgorithm "SHA1"^^xsd:string .
```

Should we create premis:Agent records for the agents?

```
<event1> a prov:InstantaneousEvent, premis:Event ;
  premis:hasEventRelatedAgent <agent1> ;

<agent1> a premis:Agent ;
  premis:agentType <http://id.loc.gov/vocabulary/preservation/agentType/sof> ;
  foaf:name "Client Software v1.2.3"^^xsd:String ;
  prov:actedOnBehalfOf <agent2> .

<agent2> a premis:Agent ;
  premis:agentType <http://id.loc.gov/vocabulary/preservation/agentType/per> ;
  foaf:nick "jquser"^^xsd:String .
```