Event Repository

Purpose

In order to be able to retrieve event histories, we maintain a persistent store of events received over the event bus linking clients, services and containers. This store is provided with a simple (Axis) web-service façade (no authentication required).

The position of this repository service can be seen in the events & messaging diagram as the "Simple Event Store"; the implementation of the service subscribes to the bus and stores all events sent over it.

A special query object is used to retrieve sets of events. There are also ports on the service for removing matching events and for writing out matching events to a log file.

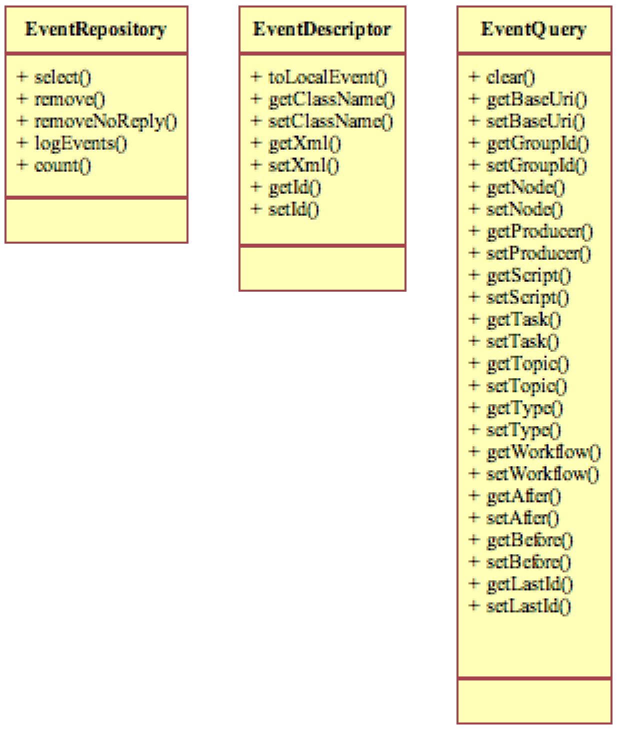
Mode of Usage

In most cases, the user will interact with this service through specially designed Siege views (refer to the tutorial for specifics). Programmatic interactions also take place via the web service ports from the Workflow Broker when cleanup or archive routines are called. A brief description of the API follows.

API

Service WSDL

EventRepository



```
public interface EventRepository
{
   public EventDescriptor[] select( EventQuery query );
   public EventDescriptor[] remove( EventQuery query );
   public void removeNoReply( EventQuery query );
   public String logEvents( EventQuery query );
   public long count( EventQuery query );
}
```

`removeNoReply` is useful when cleaning up the database (returning all events will with a high probability generate an out-of-memory error in the client).

The return value of the `logEvents` method is the URL where the log file may be found.

EventDescriptor

```
public class EventDescriptor
{
   public LocalEvent toLocalEvent() throws DeserializationException,
      UnknownExtensionException, InstantiationException,
      IllegalAccessException, ClassNotFoundException;
   public String getClassName();
   public void setClassName( String className );
   public String getXml();
   public void setXml( String xml );
   public Integer getId();
   public void setId( Integer id );
}
```

A simple wrapper around the `LocalEvent` object. Class name identifies the type of local event (necessary for deserialization).

EventQuery

```
public class EventQuery
{
   public void clear();
   public String getBaseUri();
   public void setBaseUri( String baseUri );
   public String getGroupId();
   public void setGroupId( String group );
   public String getNode();
   public void setNode( String node );
   public String getProducer();
   public void setProducer( String producer );
   public String getScript();
   public void setScript( String script );
   public String getTask();
   public void setTask( String task );
   public String getTopic();
   public void setTopic( String topic );
   public String getType();
   public void setType( String type );
   public String getWorkflow();
   public void setWorkflow( String workflow );
   public Long getAfter();
   public void setAfter( Long after );
   public Long getBefore();
   public void setBefore( Long before );
   public Integer getLastId();
   public void setLastId( Integer lastId );
}
```

A simplified version of the `LocalEventFilter`: it corresponds to a single `ncsa.tools.events.types.filters.LocalEventHeaderComparator`, a conjunction of comparisons on the header properties represented by the set- and get- methods. The `lastId` field is used by the service client for more scalable pulls (there is a default batch size of 100 events at a time).

A subclass of `EventQuery`, `ncsa.services.events.utilities.EventQueryAdapter`, allows for conversion between this object and a `LocalEventFilter` as well as providing several flags useful for interaction with the service (for instance, it is often the case that one will wish to receive all events generated up to this point in time, but to continue listening for incoming events matching that filter as well); this utility class, in fact, is used under the covers by several Siege views.

```
public class EventQueryAdapter extends EventQuery
{
   public EventQuery toQuery();
   public LocalEventFilter toFilter();
   public boolean isGetHistory();
   public void setGetHistory( boolean getHistory );
   public boolean isKeepListening();
   public void setKeepListening( boolean keepListening );
   public String getEventRepositoryUrl();
   public void setEventRepositoryUrl( String eventRepositoryUrl );
   public boolean isRemoveEvents();
   public void setRemoveEvents( boolean removeEvents );
   public GSSCredential getCredential();
   public void setCredential( GSSCredential credential );
}
```