Time Tracker Audit 3.2 Component Specification

Changes for version 3.2 are in red.

1. Design

Within the context of a large application, an *Audit Trail* is a chronological record of events that take place, to allow administrators to look back into the application's history to see what lead to the current state of affairs.

The Time Tracker Audit component handles all business and persistence logic of audit records that are created within the application. Each audit relates to a single database table row action, consisting of metadata describing the action as well as multiple 'details' with information on what information is being changed. The component itself provides a manager to allow the audit records to be added and removed, as well as searched against criteria.

This is achieved by providing the manager with a pluggable persistence layer from which information is gathered or stored. As an example of persistence, this component provides an implementation which uses an Informix database as storage.

In addition, this component defines the basic data structures for each audit header and details, as well as the enumeration of all application areas from which the audit record can be generated.

Finally, the audit searching is performed by using a set of provided search 'filters', which allow searches to be constrained by ID or Date matches, and combined through basic AND / OR / NOT operators by use of the Search Bundle component, and all the filters it provides.

Version 3.2:

As a result of the fine grain components used in this design there needs to be a transaction management strategy, which allows a single transaction to exist that encompasses all components called for a single use case. Since this component will be deployed into an Enterprise Java Bean container, JBoss 4.0.x, a Stateless Session Bean will be used to manage the transaction. The container will start a transaction when a method is invoked if one is not already running. The method will then join the new or existing transaction.   Transaction Management will be Container Managed.

In order to accommodate the new requirements, the following changes are necessary:

1. A POJO Delegate and a Stateless Session Bean are added. These provide the functionality provided by AuditManager in the former design, except that the Session Bean relies on container managed transactions.
2. The existing AuditManager class is removed from the design, replaced by an AuditManager interface that provides the same set of methods. This interface is implemented by AuditDelegate, AuditSessionBean, and AuditLocalObject.
3. The deployment descriptor will need to set the transaction level for EntrySessionBean to REQUIRED.
4. Since the transaction is now handled by the EJB container, transaction management is removed from AuditPersistence.

The new AuditSessionBean will rely on an internal AuditPersistence to handle all of its operations. It will configure this DAO using environment variables set in the deployment descriptor.

The new AuditDelegate class, which provides an interface to the AuditSessionBean and to the component as a whole, will be configured through ConfigurationManager. This is possible since it isn't run in the context of the SessionBean.

.

1.1 Design Patterns

Builder: The persistence plug-in builds audit details and headers from rows of values in database tables.
Prototype: The DefaultValuesContainer is built as a prototype AuditDetails object, from which other objects inherit missing values.
Business Delegate: The AuditDelegate class functions as a business delegate that delegates all operations to a SessionBean.
Strategy: AuditPersistence forms a strategy with AuditSessionBean where the implementation of that interface may be freely substituted through configuration.
DAO Pattern: used to abstract the persistence layer from the rest of the application
J2EE Business Interface: AuditLocalObject and AuditSessionBean both extend AuditManager, which acts as a J2EE business interface

1.2 Industry Standards

SQL is used by the provided Informix persistence layer to persist information to the database connected.

1.3 Required Algorithms

Most of the algorithms in this component come from the SQL interaction with the database - these are detailed within sequence diagrams and should be clarified by reading the suggested SQL strings provided. The other complex algorithms are from the filters - so below are two suggested pseudo code segments from AND filter:

```
String toJDBCString(){
    String ret = targets.get(0).toJDBCString();
    for(int i = 1; i < targets.size(); i++)
        ret = ret + “ AND “ + targets.get(i).toJDBCString();
    return “( “ + ret + “ )”;
}
```

```
void insertJDBCParameters(PreparedStatement stmt, int from){
    for(int i = 1; i < targets.size() && from != -1; i++)
        from = targets.get(i).insertJDBCParameters(stmt, from);
    return from;
}
```

1.3.2 Container managed transactions

Transactions will be handled by the EJB container. The AuditSessionBean will call sessionContext.setRollbackOnly() any time an operation throws an exception. It will be necessary to set the transaction level of the AuditSessionBean to REQUIRED in the deployment descriptor.

1.4 Component Class Overview

~~**AuditManager:**~~
~~AuditManager is a delegate class that allows the audits to be managed through using a pluggable persistence layer. Three of the CRUD operations are presented:~~
~~1) Create an audit, through calling the createAuditRecord member. This adds the audit header and details into persistence.~~
~~2) Retrieve specific audits according to a given search filters, by calling searchAudits~~
~~3) Delete an audit, by calling the rollbackAuditRecord member. This removes the audit header and details from persistence.~~
~~By design, there is no way to update audits once they have been inserted.~~

AuditManager <<interface>>:
This interface defines the common contract that AuditDelegate, AuditSessionBean, and AuditLocalObject all implement. These common methods are the main public functionality of the component.
The methods defined by allow a user to audit database operations, rollback audits, and to find existing audits.
Implementations of this interface are required to be thread safe.

AuditHeader:
AuditHeader is a simple bean class for the storage of information about changes to details in the Time Tracker application. It stores its unique ID, as well as multiple id values for items it is related to, as well as creation data, linked names, the area of the application that it is related to, the table of information being changed, the action performed, and the values changed. As a bean class, it provides a no-argument constructor, as well as setXXX/getXXX methods named for each member. By design, this class performs no error checking on parameters in setXXX methods; it is left to the handling application to define what constraints are put on the bean's members.

AuditDetail:
AuditDetail is a simple bean class for the storage of information about a single column change. It stores its unique ID, as well as the name of the column which was changed, and the value in the column before and after the change. As a bean class, it provides a no-argument constructor, as well as setXXX/getXXX methods named for each member. By design, this class performs no error checking on parameters in setXXX methods; it is left to the handling application to define what constraints are put on the bean's members.

ApplicationArea << enum >>:
Extending the Type Safe Enumeration component's Enum class, this enumeration contains information about all the application areas available for audits. Each

enumeration is identifiable by its name (e.g. "Expense" / "Fixed Billing"...) or its cardinal ID. ApplicationArea has a private constructor, instances are available only as public static final members of the class - in addition, each is immutable after construction.

AuditPersistence << interface >>:
Interface defining the required methods for any persistence layer to be plugged into the AuditManager. The three methods have the same definitions as the Create / Retrieve / Delete methods within the manager. All implementations of this interface should handle their own thread safety, and must also be able to be constructed through using the Object Factory - the intention is that they never need to be constructed directly.

InformixAuditPersistence:
Default Persistence plug-in provided with this component, which uses an Informix Database for storage. It implements the three required methods from the AuditPersistence interface it implements. In addition to this, two private utility methods are provided to help loading information, which can be used and modified at the discretion of the developer. This class also stores a database connection factory and connection name to provide access to a configurable database connection, as well as creating a default values contain on construction, used when loaded audit headers have missing information. Finally, an ID generator is used to obtain new IDs for audit headers and details when inserted, and a Search Bundle is kept to allow simple searching of the audits against matching filters.
In version 3.2, transaction management is removed from this class.

DefaultValuesContainer:
This package-level container is provided to allow easy storage of default values for Audit Detail records. At first, it extends the AuditDetail object, defaulting all values to null/-1 as shown on the class diagram. It is constructed by taking a configuration namespace, which stores information about any default values to override. Then, for each member in configuration, the new default value is loaded, and the appropriate setXXX is called. Afterwards, whenever an AuditDetail is loaded from persistence, if any records are null the corresponding setXXX(defaultValues#getXXX()) can be called to replace it with the desired default value.
In version 3.2, the constructor that takes a ConfigManager namespace is replaced with one that provides programmatic configuration.

com.topcoder.timetracker.audit.ejb
AuditDelegate
This is a Business Delegate/Service Locator that may be used within a J2EE application. It is responsible for looking up the local interface of the AuditSessionBean, and delegating any calls to the bean.
This class is thread safe, since all state is modified at construction.

AuditSessionBean
This is a Stateless SessionBean that is used to provided business services to manage auditing within the Time Tracker Application. It implements the AuditManager interface and delegates to an instance of AuditPersistence. Transactions for this bean are handled by the EJB Container. It is expected that the transaction level declared in the deployment descriptor for this class will be REQUIRED.
All method calls on methods in AuditManager interface are delegated to an instance of CutoffTimeDao throug the getDao() method. The getDao() method will return the value of the dao field if it is not null, or call the instantiateDao() method to create an appropriate object from configuration in the deployment descriptor if it is null.
All configuration for this class will be read from <env-entry> tags in the deployment descriptor for this bean.

The AuditManager interface implementations are required to be thread-safe, and so this stateless session bean is thread-safe also.

AuditLocalObject
Local interface for AuditSessionBean. It contains exactly the same methods as AuditManager interface.
Implementation will be generated by EJB container and thread-safety is dependent on the container.

AuditLocalHome
LocalHome interface for the AuditSessionBean. It contains only a single no-param create method that produces an instance of the local interface. it is used to obtain a handle to the Stateless SessionBean.
Implementation will be generated by EJB container and thread-safety is dependent on the container.

1.5 Component Exception Definitions

AuditManagerException:
An exception thrown by the manager whenever there are any problems in any of its methods throw during their delegation to the persistence layer. Two standard constructors are provided, both which take a message detailing why the exception was thrown, and one taking also the exception which caused it, for example which may be an AuditPersistenceException.

AuditPersistenceException:
An exception thrown by any persistence plug-in implementing the AuditPersistence, whenever there are any problems in persisting information. Two standard constructors are provided, both which take a message detailing why the exception was thrown, and one taking also the exception which caused it, for example if triggered by an SQL or IO Exception.

AuditConfigurationException:
An exception thrown by classes within this component whenever there are problems in initialization from configuration, such as missing or invalid configuration parameters. Two standard constructors are provided, both which take a message detailing why the exception was thrown, and one taking also the exception which caused it, for example which may be an Exception from the configuration manager.

1.6 Thread Safety

The Java beans within the component (AuditHeader and AuditDetails) are not threadsafe - there is no protection against concurrent read/writes of the members. Due to their intended usage, it is assumed that they will be used in a thread-safe manner, synchronization is not performed in the interest of speed and simplicity.
All implementations of the persistence interface should handle their thread safety, and the provided Informix version does this by having immutable members (log and connection name) read on construction, and not modifying the remaining members.
The Enumerations are immutable so thread-safe, and finally the AuditDelegate is thread safe as it is immutable after configuration. AuditLocalObject and AuditLocalHome implementations will be generated by the EJB Container, which will ensure their thread safe use. AuditSessionBean is not inherently thread safe, but will be managed in a thread safe manner by the EJB Container - no more than one request will be handled by any one instance concurrently.

2. Environment Requirements

2.1 Environment

- Java 1.4 for development

- Java 1.4 and 1.5 for compilation

2.2 TopCoder Software Components

- Base Exception 1.0 for the Base Exception superclass of custom exceptions.
- Configuration Manager 2.1.5 for reading parameters from configuration.
- Database Abstraction 1.1 for retrieving search results.
- DB Connection Factory 1.0 for configurable Connection loading.
- ID Generator 3.0 for help generating header and detail IDs.
- Logging Wrapper 1.2 for access to configurable Log instances.
- Object Factory 2.0.1 for loading of the pluggable persistence layer.
- Search Builder 1.3.1 for simple criteria-based searching.

2.3 Third Party Components

None

3. Installation and Configuration

3.1 Package Name

com.topcoder.timetracker.audit
com.topcoder.timetracker.audit.persistence

3.2 Configuration Parameters

Parameter	Description	Values
~~objectFactoryNamespace (required)~~	~~Namespace where the object factory's configuration is stored.~~	~~Valid Object Factory namespace.~~
~~persistenceObject (required)~~	~~Object factory key used to load the AuditPersistence implementation.~~	~~Valid Object Factory persistence key.~~
persistenceNamespace/ *connectionName* (optional)	Name of connection to acquire from DB Connection Factory - if not provided, a default connection is used.	Valid Connection Factory connection name.
persistenceNamespace/ *logName* (optional)	Name of log to acquire from the Log Factory - if not provided, a default log is used.	Valid Log Factory log name.
persistenceNamespace/ *idGeneratorName* (required)	Name of ID generator to acquire from the ID Generator factory.	Valid ID Generator factory name.
persistenceNamespace/ *searchBundleName* (required)	Name of Search Bundle to acquire from the Search Bundle Manager.	Valid Search Bundle factory name.
persistenceNamespace/ defaultValueNamespace (required)	Namespace used to load default Audit Detail values from	Valid Configuration namespace.
defaultValues/ id (optional)	Default value to use for new IDs	Parseable Long
defaultValues/ newValue (optional)	Default value to use for Audit Details with no new value.	Any String
defaultValues/ oldValue (optional)	Default value to use for Audit Details with no old value.	Any String
defaultValues/ columnName (optional)	Default value to use for Audit Details with no column Name.	Any String.

The following properties may be specified to the AuditDelegate class through Configuration Manager.

context_name	This is the context name used to retrieve the home object of the respective session bean.	Optional; If not specified, then the default context provided by JNDIUtils is used	Any valid JNDI context.
jndi_home	This is the name used to retrieve the EJBLocalHome object for AuditSessionBean	Required	Valid name for AuditLocalHome object in context

For AuditSessionBean - these will be read from com.topcoder.timetracker.audit.ejb.AuditSessionBean:

of_namespace	The namespace used for ConfigManagerSpecificationFactory to use with ObjectFactory	Required	A valid object factory namespace
dao_key	The key for an AuditPersistence	Required	An object factory

	object to get from ObjectFactory		key referring to a NotificationManager

3.3 Dependencies Configuration

All components listed in 2.2 (except BaseException) will require some form of configuration to allow the configuration of *this* component to be used - e.g. connections and logs set up. Please see their component specification for more details.

4. Usage Notes

4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute ‘ant test’ within the directory that the distribution was extracted to.

4.2 Required steps to use the component

See below for example usage:

4.3 Demo

The following demo has been adapted for the use of AuditDelegate rather than the old AuditManager class for version 3.2.

```
// assuming a valid configuration file is provided in the ConfigManager
// namespace com.topcoder.timetracker.audit
AuditDelegate delegate = new AuditDelegate("com.topcoder.timetracker.audit);
// get some audit record (i.e. generated from insert or web form)
AuditHeader record = ...;
delegate.createAuditRecord(record); // added to Informix persistence
assert(record.isPersisted());      // assume no problems yet

// add record with failure:
try{
    delegate.createAuditRecord(record);
} catch(AuditManagerException e){
    // should be thrown, due to duplicate id. Check:
    assert(record.isPersisted() == false); // not persisted:
    // the record’s id and detail ids should be logged
}

// search audit headers for the previously added record:
Integer refId = new Integer(record.getId());
Filter filter = SearchBuilder.buildEqualToFilter(“audit_id”, refId);
AuditHeader[] matches = delegate.searchAudit(filter);
assert(matches.length == 1); // only one can match a given audit id

// check it’s the same:
AuditHeader found = matches[0];
assert(found.getId() == record.getId());
assert(found.getApplicationArea() == record.getApplicationArea());

// remove:
boolean removed = delegate.rollbackAuditRecord(record.getId());
assert(removed == true); // header and details removed

// search audit headers again:
AuditHeader[] matches = delegate.searchAudit(filter);
assert(matches.length == 0); // no match now it has been removed

// remove again:
boolean removed = delegate.rollbackAuditRecord(record.getId());
assert(removed == false); // if no record exists, false is returned.
```

5. Future Enhancements

One possible area of improvement would be in allowing new ApplicationAreas to be added programmatically. Currently, the a new enumeration value must be added, the class recompiled, and a new row added to the database - while simple, it does not support changes while the application is running.

Another enhancement that may be added is to provide more information about what happened when persistence failed - the isPersisted flag could be replaced, say with a listener method that is called whenever a persistence action fails.

Finally, either a generic Time Tracker search criteria component could be used to replace the filters within this component, or the SQL builder component could be used provided it is as simple as these filters to use.

6. Appendix:

The Entity Relationship diagram has been provided with the submission.
The following is a list mapping Audit Header members to audit table columns:

Member	Column
`id`	`audit_id`
`entityId`	`entity_id`
`creationDate`	`creation_date`
`tableName`	`table_name`
`companyId`	`company_id`
`creationUser`	`creation_user`
`actionType`	`action_type`
`clientId`	`client_id`
`projectId`	`project_id`
`resourceId`	`account_user_id`
`applicationArea (using its getId())`	`app_area_id`

The clientName and projectName are on external tables, matching their respective IDs.

Also, the application area enumerations will have to exist within the application_area table, with whatever creation/modification values are required, and an ID equivalent to their getId() value - 0 for TT_EXPENSE, 1 for TT_FIXED_BILLING etc.