Time Tracker Fixed Billing Entry version 3.1 Component Specification

1.Design

The Time Tracker Fixed Billing Entry custom component is part of the Time Tracker application. It provides an abstraction of Fixed Billing Entries. This component handles the persistence and other business logic required by the application.

The design is separated into 2 layers of management: The topmost layer are the Manager classes, which provide the functionality needed to update, retrieve and search from the data store. Batch operations are also available in the Manager classes. The next layer are the DAOs, which interact directly with the data store. There is also a sublayer, called the FilterFactory layer, which are responsible for building search filters which can be used to build the search filter.

There is an additional J2EE layer on top of the component. This layer is composed of a SessionBean and a BusinessDelegate. The SessionBean is stateless, and supports local interfaces. The BusinessDelegate doubles as a service locator, and looks up the Session Bean upon construction, delegating all similar method calls to the SessionBean. This layer is responsible for transactions, allowing a J2EE container to manage transactions for the component.

1.1Design Patterns

The Strategy Pattern. *The DAO classes such as FixedBillingEntryDAO and FixedBillingStatusDAO are strategies for persisting and retrieving information from a data store.*

The Facade Pattern. *The manager classes [FixedBillingEntryManager, FixedBillingStatusManager] encapsulate subsystem DAOs and other components to provide a unified API that makes it easier to manage the entries.*

The Factory Pattern. *The Factory pattern is used in the FilterFactory layer to create search filters. The FixedBillingEntryFilterFactory and FixedBillingStatusFilterFactory employ this pattern.*

The Business Delegate Pattern. *The Business Delegate pattern is used with the FixedBillingEntryDelegate and FixedBillingStatusDelegate classes.*

The Service Locator Pattern. *The above-mentioned classes also double as Service Locators.*

1.2Industry Standards

JDBC 3.0

EJB 2.1

1.3Required Algorithms

There were no algorithms required and the component is straightforward enough. We will discuss the database schema, and the method of searching in this section.

1.3.1 Data Mapping

This section will deal with mapping the different values in the beans to their respective columns. The developer is responsible for generating the necessary SQL to insert/retrieve the data in the beans from the appropriate columns. The SQL will require some simple table joins at the most. Please refer to the ERD diagrams for more information [see TimeTrackerFixedBillingEntry_ERD.jpg].

1.3.1.1 FixedBillingEntry Class and fix_bill_entry Table

- Column fix_bill_entry_id maps to the id property in the TimeTrackerBean base class.
  Note: The id generator is used to create a new id when a new record is inserted.
- Column company_id maps to the companyId property in the BaseEntry superclass.
- Column fix_bill_type_id does not map to anything [it was clarified in the forums that it was not needed]
- Column fix_bill_status_id maps to the FixedBillingStatus property in the FixedBillingEntry class.
- Column description maps to the description property in the FixedBillingEntry class.
- Column entry_date maps to the entryDate property in the BaseEntry superclass.
- Column amount maps to the amount property in the FixedBillingEntry class.
- Column creation_date maps to the creationDate property in the TimeTrackerBean base class.
- Column creation_user maps to the creationUser property in the TimeTrackerBean base class.
- Column modification_date maps to the modificationDate property in the TimeTrackerBean base class.
- Column modification_user maps to the modificationUser property in the TimeTrackerBean base class.
- Column invoice_id maps to the invoice id property in the FixedBillingEntry class.

1.3.1.2 FixedBillingStatus Class and fix_bill_status Table

- Column fix_bill_status_id maps to the id property in the TimeTrackerBean base class.
  Note: The id generator is used to create a new id when a new record is inserted.
- Column description maps to the description property in the FixedBillingStatus class.
- Column creation_date maps to the creationDate property in the TimeTrackerBean base class.
- Column creation_user maps to the creationUser property in the TimeTrackerBean base class.
- Column modification_date maps to the modificationDate property in the TimeTrackerBean base class.
- Column modification_user maps to the modificationUser property in the TimeTrackerBean base class.

1.3.2 Searching

Searching is executed in this component by using the Search Builder component and the FilterFactory layer. First off, the developer needs to develop a search query

off which to base the search on. The search query will retrieve all the necessary attributes, and join with the tables of any possible criterion. Once a query has been defined, in the constructor, the appropriate FilterFactory will be initialized with the column names used in the query. This FilterFactory can then be returned by the appropriate getFilterFactory method.

The user may then use the Factory to build the search filters. Once the filters are provided back to the DAO implementation, it may use the DatabaseSearchStrategy to build the search. The DatabaseSearchStrategy will then add the necessary WHERE clauses to constrain the search to the search criterion specified in the Filters.

Example:

The following query may be used as the context for searching the FixedBillingEntries(we reduce the retrieved data to only the id, but developer may modify it to retrieve all relevant fields):

```
SELECT
      fix_bill_entry_id
FROM
      fix_bill_entry
INNER JOIN
      fb_reject_reason
      ON
      fix_bill_entry.fix_bill_entry_id =
fb_reject_reason.fix_bill_entry_id
WHERE
```

From this context, the Search Builder can then add the different WHERE clauses, depending on the filter that was provided. For example, if a filter for the description was created using createCreationDateFilter, then the Search Builder would add:

```
WHERE (continued from above)
      fix_bill_entry.creation_date = [value of filter]
```

To accomplish this, in the constructor of the DAO, the FilterFactory must be configured according to the context query that was used. In this example, the mapped value for the CREATION_DATE_COLUMN_NAME should be "fix_bill_entry.creation_date".

Finally, note that this may be optimized by adding a GROUP BY clause to group the results by ids (this is especially important if a reject reason filter is not specified, since it would return multiple rows for each reject reason in a given filter). The GROUP BY clause may be manually added by calling the buildSearchContext method, and adding the GROUP BY clause after the buildSearchContext has been completed. This will need to be done in a private inner subclass of SearchStrategy.

1.3.3 Auditing

The method implementation notes contain the audit header information that should be used when performing the audit. The following clarifies how to create AuditDetail objects to add to the AuditHeader. When creating a new entry (audit header action type is CREATE),

then the oldValue for each detail is null. When deleting an entry (audit header action type is DELETE), then the newValue for each detail is null. When updating an entry, the old value is retrieved from the datastore to populate the AuditDetail's old value.

1.3.4 Batch Operations

The DbDAO classes have a useBatching variable assigned to them. If this is true, then the developer should use the *addBatch* method with PreparedStatements when performing batch operations. All batch methods in the DAO should be atomic, and so if a BatchUpdateException occurs, then the entire batch must be rolled-back, to ensure consistency between different JDBC drivers. Note that batching is not required for DELETE statements, since an IN clause may be used for that.

1.3.5 Configuration via Object Factory

The component relies on the Object Factory for configuration. A sample configuration file has been provided in the docs directory. Initializing the Manager classes simply consists of calling the *createObject* method with the appropriate class argument.

1.4Component Class Overview

Package com.topcoder.time.tracker.entry.fixedbilling

FixedBillingEntryManager (interface)

This interface represents the API that may be used in order to manipulate the various details involving a Time Tracker Fixed Billing Entry. CRUDE and search methods are provided to manage the Time Entries inside a persistent store. There are also methods for the manipulation of RejectReasons associated with the FixedBillingEntry.

FixedBillingEntryDAO (interface)

This is an interface definition for the DAO that is responsible for handling the retrieval, storage, and searching of Time Tracker FixedBillingEntry data from a persistent store. It is also responsible for generating ids for any entities within it's scope, whenever an id is required.

FixedBillingEntryFilterFactory (interface)

This interface defines a factory that is capable of creating search filters used for searching through Time Tracker FixedBillingEntries. It offers a convenient way of specifying search criteria to use. The factory is capable of producing filters that conform to a specific schema, and is associated with the FixedBillingEntryManager that supports the given schema.

BaseFilterFactory (interface)

This is a base FilterFactory interface that provides filter creation methods that may be used for filters of any Time Tracker Bean. It encapsulates filters for the common functionality - namely, the creation and modification date and user.

FixedBillingEntryRejectReasonDAO (interface)

This is an interface definition for the DAO that is responsible for handling the association between Time Tracker FixedBillingEntries and the Reject Reasons. Simple CRUDE methods are specified.

FixedBillingStatusManager (interface)

This interface represents the API that may be used in order to manipulate the various details involving a FixedBillingStatus. CRUDE and search methods are provided to manage the FixedBillingStatuses inside a persistent store.

FixedBillingStatusDAO (interface)

This is an interface definition for the DAO that is responsible for handling the

retrieval, storage,and searching of FixedBillingStatus data from a persistent store. It is also responsible for generating ids for any entities within it's scope, whenever an id is required.

FixedBillingStatusFilterFactory (interface)

This interface defines a factory that is capable of creating search filters used for searching through FixedBillingStatuses. It offers a convenient way of specifying search criteria to use. The factory is capable of producing filters that conform to a specific schema, and is associated with the FixedBillingStatusManager that supports the given schema.

StringMatchType

This is an enum that is used by the FilterFactories for those methods when a String criterion is specified. This gives the user a convenient way of specifying the method of matching the Strings.

FixedBillingEntry

This is the main data class of the component, and includes getters and setters to access the various properties of a Fixed Billing Entry, A Fixed Billing Entry is an amount of money spent by a Project Manager for a specific Project and client.

FixedBillingStatus

This is a bean that represents a FixedBillingEntryStatus, which represents a possible state that a FixedBillingEntry can have in the context of the Time Tracker system.

FixedBillingManagerImpl

This is a default implementation of the FixedBillingEntryManager interface. it utilizes instances of the FixedBillingEntryDAO in order to fulfill the necessary CRUDE and search operations defined for the Time Tracker FixedBillingEntries. It also uses FixedBillingEntryRejectReasonDAO to associate any RejectReasons with the FixedBillingEntries.

FixedBillingStatusManagerImpl

This is a default implementation of the FixedBillingStatusManager interface. it utilizes instances of the FixedBillingStatusDAO in order to fulfill the necessary CRUDE and search operations defined for the Time Tracker FixedBillingStatuses.

ManagerFactory

This is a class that acts as a factory for the managers, and may be used to easily access the manager class for various purposes. It uses lazy instantiation to create the managers. Instantiation is done through the Object Factory.

Package com.topcoder.time.tracker.entry.fixedbilling.db

BaseDAO

This is a base DAO class that encapsulates the common elements that may be found within a DAO such as the connection details, id generator, search strategy and audit manager.

DbFixedBillingEntryDAO

This is an implementation of the FixedBillingEntryDAO interface that utilizes a database with the schema provided in the Requirements Section of Time Tracker Fixed Billing Entry 3.1.

DbFixedBillingStatusDAO

This is an implementation of the FixedBillingStatusDAO interface that utilizes a database with the schema provided in the Requirements Section of Time Tracker Fixed Billing Entry 3.1.

DbFixedBillingEntryRejectReasonDAO

This is an implementation of the DbFixedBillingEntryRejectReasonDAO interface that utilizes a database with the schema provided in the Requirements Section of Time Tracker Fixed Billing Entry 3.1.

Package com.topcoder.time.tracker.entry.fixedbilling.filterfactory

MappedBaseFilterFactory

This is an implementation of the BaseFilterFactory that may be used to build filters.

MappedFixedBillingStatusFilterFactory

This is an implementation of the FixedBillingStatusFilterFactory that may be used to build filters.

MappedFixedBillingEntryFilterFactory

This is an implementation of the FixedBillingEntryFilterFactory that may be used to build filters.

Package com.topcoder.time.tracker.entry.fixedbilling.filterfactory

FixedBillingEntryManagerDelegate

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 SessionBean for FixedBillingEntryManager, and delegating any calls to the bean.

FixedBillingStatusManagerDelegate

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 SessionBean for FixedBillingStatusManager, and delegating any calls to the bean.

FixedBillingEntrySessionBean

This is a Stateless SessionBean that is used to provided business services to manage FixedBillingEntries within the Time Tracker Application. It contains the same methods as FixedBillingEntryManager, and delegates to an instance of FixedBillingEntryManager.

FixedBillingStatusSessionBean

This is a Stateless SessionBean that is used to provided business services to manage FixedBillingStatuses within the Time Tracker Application. It contains the same methods as FixedBillingStatusManager, and delegates to an instance of FixedBillingStatusManager.

FixedBillingEntryManagerLocal

Local interface for FixedBillingEntryManager. It contains exactly the same methods
as FixedBillingEntryManager interface.

FixedBillingStatusManagerLocal

Local interface for FixedBillingStatusManager. It contains exactly the same methods
as FixedBillingStatusManager interface.

FixedBillingEntryManagerLocalHome

LocalHome interface for the FixedBillingEntryManager. 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.

FixedBillingStatusManagerLocalHome

LocalHome interface for the FixedBillingStatusManager. 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.

1.5Component Exception Definitions

DataAccessException

This exception is thrown when a problem occurs while this component is interacting with the persistent store. It is thrown by all the DAO and Manager interfaces (and their respective implementations).

UnrecognizedEntityException

This exception is thrown when interacting with the data store and an entity cannot be recognized. It may be thrown when an entity with
a specified identifier cannot be found. It is thrown by all the Manager and DAO interfaces (and their implementations).

InvalidCompanyException

This exception is thrown when there is an attempt to relate a Time Tracker entities to a FixedBillingEntry whose companies do not match. For example, adding a RejectReason to a FixedBillingEntry when their companies are different.

ConfigurationException

This exception is thrown when there are problems with configuration.

InvalidFilterException

This exception is thrown when there is request to perform a search with filters that cannot be recognized by the given DAO or Manager class.

BatchOperationException

This exception is thrown during batch operations where one or more of the elements in the batch failed to be processed properly. It will contain an array of causes: each element that managed to be processed successfully will have a null for the corresponding element in the causes array, and each element that failed to be processed will have an Exception detailing the reason for failure in the causes array.

1.6Thread Safety

This component is not completely thread-safe, The Bean instances are not thread safe, and it is expected to be used concurrently only for read-only access. Otherwise, each thread is expected to work with its own instance.

The Manager classes are required to be thread safe, and achieves this via the thread safety of the implementations of the DAO Layer, and the FilterFactory Layer.

The DAOLayer is made thread safe through the use of transactions, and is achieved with the transaction level of READ_COMMITTED.

The FilterFactory layer is thread-safe by virtue of being immutable.

2.Environment Requirements

2.1Environment

Java 1.4

2.2TopCoder Software Components

DB Connection Factory 1.0 – for creating the DB connections. The Connection Factory may be configured to retrieve connections from the JBoss TXDataSource if desired.

Base Exception 1.0 – base class for custom exception is taken from it

Object Factory 2.0.1 – is indirectly used to configure the component.

ID Generator 3.0 – for generating IDs in the persistence implementation.

TypeSafe Enum 1.0 – is used for the StringMatchType enum.

Search Builder 1.3.1 – is used to perform the searches

JNDI Context Utility 1.0 - is used by the business delegates to look up the home interface of the session bean.

ConfigManager 2.1.5 – was not used because it relies on Object Factory for configuration.

Time Tracker Audit 3.1 – is used to perform the optional audit.

Time Tracker Base Entry 3.1 – is used to provide the base entry class.

Time Tracker Reject Reason 3.1 – is used to retrieve the RejectReason information.

Time Tracker Common 3.1 – is used to provide the TimeTrackerBean base class.

2.3Third Party Components

None

3.Installation and Configuration

3.1Package Names

com.topcoder.time.tracker.entry.fixedbilling
com.topcoder.time.tracker.entry.fixedbilling.db
com.topcoder.time.tracker.entry.fixedbilling.filterfactory
com.topcoder.time.tracker.entry.fixedbilling.j2ee

3.2Configuration Parameters

This component relies on Object Factory 2.0 for configuration that is based from a file. See the sample configuration file (docs/Time_Tracker_Fixed_Billing_Entry_Object_Factory_Sample_Config.xml) provided on how to configure Object Factory to do this.

For the j2ee portion, the following config parameters are used by all business delegates:

For FixedBillingEntryManagerDelegate and FixedBillingStatusManagerDelegate:

Property Name: *contextName*
Property Description: This is the context name used to retrieve the home object of the

respective session bean. If not specified, then the default context provided by JNDIUtils is used.
Is Required: *Optional*

3.3Dependencies Configuration

All the dependencies are to be configured according to their component specifications.

Do note that the default configuration for Search Builder is desired to be able to build the search string correctly.

4.Usage Notes

4.1Required steps to test the component

Extract the component distribution.

Follow Dependencies Configuration.

Execute 'ant test' within the directory that the distribution was extracted to.

4.2Required steps to use the component

Configure the dependency components.

4.3 Demo

We will assume here that everything is configured properly. Try/catch clauses have been removed to enhance clarity.

```
// Create a FixedBillingEntryManagerDelegate
FixedBillingEntryManager entryManager = new
FixedBillingEntryManagerDelegate("applicationNamespace");

//  4.3.1 Creating a new Entry
FixedBillingEntry newEntry = new FixedBillingEntry();
newEntry.setCompanyId(tcCompanyId);
newEntry.setDescription("Dinner");
newEntry.setEntryDate(new Date());
newEntry.setStatus(pendingStatus);

// Register the entry with the manager, with auditing.
entryManager.createFixedBillingEntry(newEntry, true);

// 4.3.2 Changing the Entry Details

// Retrieve an entry from the manager
FixedBillingEntry changingEntry = entryManager.getFixedBillingEntry(entryIdToChange);

// Update the entry details.
changingEntry.setDescription("Purchase Replacement Parts");
changingEntry.setStatus(rejectedStatus);

// Update the entry in the manager
entryManager.updateFixedBillingEntry(changingEntry);
```

```
// Add a RejectReason to the Entry
entryManager.addRejectReasonToEntry(changingEntry, rejectReasonId);

// 4.3.3 Search for a group of Entries with "Purchase" in the description and status of
"pending" and delete them.  The delete operation should be atomic.

// Define the search criteria.
FixedBillingEntryFilterFactory filterFactory = entryManager.getFixedBillingEntryFilterFactory();
List criteria = new ArrayList();
criteria.add(filterFactory.createDescriptionFilter("Purchase", StringMatchType.SUBSTRING));
criteria.add(filterFactory.createStatusFilter(pendingStatus));

// Create a search filter that aggregates the criteria.
Filter searchFilter = new AndFilter(criteria);

// Perform the actual search.
FixedBillingEntry[] matchingEntries = entryManager.searchFixedBillingEntries(searchFilter);

// Delete the users; auditing is performed.
entryManager.deleteFixedBillingEntries(matchingEntries, true);

// 4.3.4 Retrieve all Reject Reasons for a specific entry and delete them; The operation
should be non-atomic.

long[] rejectReasonIds = entryManager.getAllRejectReasonsForEntry(targetEntryId);

// A for loop will be used to feed the rejectReasonIds singularly.  The application container will
take care of rolling back any failed transactions.
for (int x = 0; x < rejectReasonIds.length; x++) {
        entryManager.removeRejectReasonFromEntry(targetEntryId, rejectReasonIds[x]);
}

// 4.3.5 Create a new FixedBillingStatus

// Create a FixedBillingStatusManagerDelegate
FixedBillingStatusManager statusManager = new
FixedBillingStatusManagerDelegate("applicationNamespace");

// Define a Status
FixedBillingStatus status = new FixedBillingStatus();
status.setDescription("Tentative Approved");
status.setCreationUser("TCUSER");
status.setModificationUser("TCUSER");

// Create the status in the manager.
statusManager.createFixedBillingStatus(status);

// A slight mistake was made in the description; Change it.
status.setDescription("Tentative Approval");

// Update the status in the manager.
statusManager.updateFixedBillingStatus(status);

// Verify the status exists by searching for it.
FixedBillingStatusFilterFactory statusFilterFactory =
statusManager.createFixedBillingStatusFilterFactory();
```

```
FixedBillingStatus[] statusResults =
statusManager.searchFixedBillingStatuses(statusFilterFactory.createDescriptionFilter("Tentative
Approval", StringMatchType.EXACT_MATCH));

// 4.3.6 Batch operations (delete is already shown in 4.3.3)

//  Batch create (we assume entriesToCreate has already been defined)
entryManager.createFixedBillingEntries(entriesToCreate, true);

// Retrieve a set of entries, change the description and update them (we assume
entryIdsToRetrieve is already defined).
FixedBillingEntry[] entriesToUpdate = entryManager.getFixedBillingEntries(entryIdsToRetrieve,
true);
for (int x = 0; x< entriesToUpdate.length; x++) {
    entriesToUpdate[x].setDescription("Board and Lodging");
}


// Perform the batch update
entryManager.updateFixedBillingEntries(entriesToUpdate, true);
```

5.Future Enhancements

Provide implementations for different RDBMS.