Time Tracker Invoice version 1.0 Component Specification

1.Design

The Time Tracker Invoice custom component is part of the Time Tracker application. It provides an abstraction of an Invoice used to bill a client once time, expense and fixed billing entries are entered for a project. 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 is the Manager classes, which provide the functionality needed to update, retrieve and search from the data store. The next layer is the DAO's layer, which interact directly with the data store.

The J2EE layer, which consists of a Local, LocalHome, Delegate, and SessionBean classes for each of the different entities, is the manager layer implementation because all business logic resides in session bean. The Local and LocalHome interfaces define the methods needed by the J2EE container to create the implementing classes. The Delegate, Local and SessionBean classes all implement/extend from their respective Manager interface, and therefore have the same set of business methods. The Delegate looks up the Local interface and routes all method calls to the Local interface. The Local interface then routes those calls to the SessionBean. The SessionBean is Manager implementation and delegate the persistence logic to the dao.

Those are the features of this design:

- No more state is maintained within the SessionBeans. The SessionBean 'looks up' the relevant DAOs using a DAOFactory. The InvoiceDAOFactory acts like a singleton, and functions as a global access point for the DAO.
- A sample Deployment Descriptor is provided in the docs directory.
- A note was added to the Use Cases, saying that the application may now run under a J2EE container. the requirements for J2EE layer are not business functionality, so it did not seem appropriate to introduce an use case for the J2EE layer.
- The JBoss Transaction DataSource may be configured using ConnectionFactory.
- The design attempts to adhere to the J2EE specification of not allowing File access by delegating all file access to an external InvoiceDAOFactory class. This is a pragmatic approach. File access does not occur within the SessionBean itself, but rather within external classes when the application is called. It is also possible for File Access to occur before any SessionBean calls occur, since the ConfigManager may be initialized beforehand.
- Note that the only other alternative to not using ConfigManager and configuration files will be to restrict usage of ANY TC components that utilize configuration files - these include the Connection Factory, Search Builder and ID Generator components, which are a core part of the given component.

1.1Design Patterns

The Strategy Pattern. *The DAO class such as InvoiceDAO is a strategy for persisting and retrieving information from a data store.*

The Facade Pattern. *The InvoiceManager class encapsulates some subsystem DAOs and other components to provide a unified API that makes it easier to manage the Invoices.*

The Factory Pattern. *The Factory pattern is used in InvoiceDAOFactory.*

The Business Delegate Pattern. *The Business Delegate pattern is utilized in the InvoiceManagerDelegate class.*

The Service Locator Pattern. *The Service Locator pattern is utilized in the same classes as the Business Delegate pattern. These classes perform a dual-function as a business delegate and service locator.*

1.2Industry Standards

None

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.

1.3.1.1 Invoice Class: invoice Table [see TimeTrackerInvoice_ERD.jpg]

Column invoice_id maps to the id property in the TimeTrackerBean superclass of the Invoice class.
Note: The id generator is used to create a new invoice_id when a new record is inserted.

Column project_id maps to the projectId property in the Invoice class.

Column invoice_status_id maps to the id of the Status property in the Invoice class.

Column creation_date maps to the creationDate property in the TimeTrackerBean superclass of the Invoice class.

Column creation_user maps to the creationUser property in the TimeTrackerBean superclass of the Invoice class.

Column modification_date maps to the modificationDate property in the TimeTrackerBean superclass of the Invoice class.

Column modification_user maps to the modificationUser property in the TimeTrackerBean superclass of the Invoice class.

Column salesTax maps to the salesTax property in the Invoice class.

Column payment_term maps to the id of PaymentTerm property in the Invoice class.

Column invoice_number maps to the invoiceNumber property the Invoice class.

Column po_number maps to the purchaseOrderNumber property the Invoice class.

Column invoice_date maps to the invoiceDate property the Invoice class.

Column due_date maps to the idueDate property the Invoice class

Column paid maps to the paid property the Invoice class

1.3.1.2 InvoiceStatus Class: invoice status table [see TimeTrackerInvoice_ERD.jpg]

Column invoice_status_id maps to the id property in InvoiceStatus

Column creation_date maps to the creationDate property in InvoiceStatus

Column creation_user maps to the creationUser property in InvoiceStatus

Column modification_date maps to the modificationDate property in InvoiceStatus

Column modification_user maps to the modificationUser property in InvoiceStatus

Column description maps to the description property in the InvoiceStatus.

1.3.2 Searching

Searching is executed in this component by using the Search Builder component and. 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, the developer must define the filters creation in InformixInvoiceFilterFactory

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 appropriate SearchBundle to build the search. SearchBundle is retrieved from SearchBundleManager. SearchBundleManager defines in its namespace the search bundles. So the query and the definition of filters are in the SearchBundleManager namespace.

Example:

The following query may be used as the context for searching the Invoices (this example is for depth INVOICE_ONLY):

```
SELECT
      invoice_date ad invoice_invoice_date
FROM
      invoice
WHERE
```

From this context, the Search Builder can then add the different WHERE clauses, depending on the filter that was provide. The definitions of the filters are put below the definition of the context. See Search Builder Component for a simple queries config file.

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 you create or update an entry the operation must audit. So it must create an AuditDetail for each columns. When creating a new entry (audit header action type is INSERT), then the oldValue for each detail is null. When updating an entry (audit header action type is UPDATE), the old value is retrieved from the datastore to populate the AuditDetail's old value. The other details for audit an operation are in the documentation tab of the method.

1.3.4 Transaction control

The transaction control is completely managed by the ejb container, so the DAO must be not control the transaction.

1.3.5 Create/update an Invoice

This is the algorithm for create an invoice:

```
Get a connection
Create the correct statement for create an Invoice
Get a new id from id generator
Execute the statement
Audit the operation if the audit is turned on
Update the expense entries with this invoice id using Expense Manager
Update the service details with this invoice id using Service Detail Manager
Update the fixed billing with this invoice id using Fixed Billing Manager
```

The algorithm for update an Invoice is very similar:

```
Get a connection
Create the correct statement for update an Invoice
Execute the statement
Audit the operation if the audit is turned on
Update the expense entries with this invoice id using Expense Manager
Update the service details with this invoice id using Service Detail Manager
Update the fixed billing with this invoice id using Fixed Billing Manager
Update the Payment Term with this invoice id using Common Manager
```

The algorithms are similar because the entries are updated always. Also the Payment is updated.

1.3.5 Status of the entries

The status of the entries is retrieved with the correct manager . The manager when retrieves an entry, retrieves also its status and set it to the entry bean. So the status is checked with the getter method of the entry bean.

1.4Component Class Overview

Package com.topcoder.timetracker.invoice

Invoice

This is the main data class of the component, and includes getters and setters to access the various properties of a Time Tracker Invoice, This class encapsulates the invoice's information within the Time Tracker component.
It also extends from the base TimeTrackerBean to include the id, creation and modification details.

InvoiceStatus

This class includes getters to access the various properties of a Time Tracker Invoice Status, This class encapsulates the invoice status information within the Time Tracker component. The setters are not shown because it's an enumeration and it must not change its state. The constructor is public because in the design forum it's asked this. This class doesn't inherited from Time Tracker Bean because it's asked this in the forum and because then TT Bean have the setter methods that must not be in this class.

InvoiceManager (interface)

This interface represents the API that may be used in order to manipulate the various details involving a Time Tracker Invoice and Invoice Status. CRUD and search methods are provided to manage the Invoice inside a persistent store.

InvoiceDAO (interface)

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

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

InvoiceStatusDAO (interface)

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

InvoiceDAOFactory

This is a class that acts as a factory for the InvoiceDAO and InvoiceStatusDAO, and may be used to easily access the dao classes for various purposes. It uses lazy instantiation to create the manager implementations. Instantiation is done through the Object Factory. This is used by the SessionBean to delegate the work to.

Package com.topcoder.time.tracker.invoice.informix

InformixInvoiceDAO

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

InformixInvoiceStatusDAO

This is an interface definition for the DAO that is responsible for handling the retrieval Time Tracker Invoice Status data from a persistent store. It's assumed that the invoice statuses don't change during the application lifetime so all statuses are loaded once time, so this improves the efficiency. The only thing that changes is the status of Invoice as in the RS is specified.

InformixInvoiceFilterFactory

This class may be used for building searches in the database. It builds filters according to the specified column names. The column names are defined in the config file for Search Bundle, so the names of the columns used are related to it.
It may be possible to create complex criterion by combining the filters produced by this factory with any of the Composite Filters in the Search Builder Component (AndFilter, OrFilter, etc.)

Package com.topcoder.timetracker.invoice.ejb

InvoiceManagerLocal

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

InvoiceManagerLocalHome

LocalHome interface for the InvoiceManager. 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.

InvoiceManagerDelegate

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 InvoiceManager, and delegating any calls (and the constants) to the InvoiceManagerLocal.

InvoiceSessionBean

This is a Stateless SessionBean that is used to provided business services to manage Invoices within the Time Tracker Application. It implements the InvoiceManager interface and delegates to an instance of InvoiceDAO and an instance of

InvoiceStatusDAO retrieved from InvoiceDAOFactory.

InvoiceSearchDepth

This is the enum for the possible depths search in searchInvoices method in InvoiceDAO.

1.5Component Exception Definitions

InvoiceDataAccessException [Custom]

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).

InvoiceUnrecognizedEntityException [Custom]

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).

InvoiceConfigurationException [Custom]

This exception is thrown when there is a problem that occurs when configuring this component.

IllegalArgumentException

This exception is thrown when an argument in a method is invalid

CreateException

This exception is used as a standard application-level exception to report a failure to create an entity EJB object.

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 InvoiceManager implementation class is required to be thread safe, and achieves this via the thread safety of the implementations of the DAO Layer. Currently implementation is the Session Bean. Thread-safety of the J2EE layer is dependent on the statelessness and thread safety of the manager classes, and also on the application container controlling them. The only thing to pay attention to is that the inner state of the classes is not modified after creation.

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

2.Environment Requirements

2.1Environment

Java 1.4

2.2TopCoder Software Components

DB Connection Factory 1.0 – for creating the DB connections

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

Object Factory 2.0 – is indirectly used to configure the component.

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

Search Builder 1.3.1 – is used to perform the searches

TypeSafe Enum 1.0 - is used to enumerate the different Status of the Invoice and the different search depths

ConfigManager 2.1.5 – is used configure certain portions of the component (also used by Object Factory).

Time Tracker Audit 3.2 – is used to perform the optional audit.

Time Tracker Common 3.2 – is used to provide the TimeTrackerBean base class and PaymentTerm.

JNDI Context Utility 1.0 - for retrieving the LocalHome interface implementations from the container.

Time Tracker Expense Entry 3.2– is used to provide the ExpenseEntry class and the manager.

Time Tracker Fixed Billing Entry 3.2 – is used to provide the FixedBilling class and the manager.

Time Tracker Service Detail 3.2 – is used to provide the ServiceDetails class and the manager.

Time Tracker Project Entry 3.2 – is used to provide the Manager that it's used for lookup of all entries from a project (canCreateInvoice method).

2.3Third Party Components

None

3.Installation and Configuration

3.1Package Names

com.topcoder.timetracker.invoice
com.topcoder.timetracker.invoice.informix
com.topcoder.timetracker.invoice.ejb

3.2Configuration Parameters

This component relies on Object Factory 2.0 for configuration that is based from a file. Look the sample config file: **config.xml.**

For InformixInvoiceStatusDAO:

The InvoiceStatus utilizes the following properties for retrieve all statuses from db:

objectFactoryNamespace - This is the namespace that is used to create a ConfigManagerSpecification factory that will be used to initialize the ObjectFactory used to create the DBConnectionFactory implementation to use. **Required.**

dbConnectionFactoryKey - An identifier that is provided to the createObject method of ObjectFactory for create DBConnectionFactory. **Required.**

For InvoiceDAOFactory:

The InvoiceStatus utilizes the following properties for create the InvoiceDAO:

objectFactoryNamespace - This is the namespace that is used to create a ConfigManagerSpecification factory that will be used to initialize the ObjectFactory used to create the InvoiceDAO implementation to use. **Required.**

invoiceDAOKey - An identifier that is provided to the createObject method of ObjectFactory for create InvoiceDAO. **Required.**
invoiceStatusDAOKey - An identifier that is provided to the createObject method of ObjectFactory for create InvoiceStatusDAO. **Required.**

For InformixInvoiceDAO:

The InvoiceStatus utilizes the following properties for configuring the fields:

objectFactoryNamespace - This is the namespace that is used to create a ConfigManagerSpecification factory that will be used to initialize the ObjectFactory. **Required.**

auditManagerKey - An identifier that is provided to the createObject method of ObjectFactory for create AuditManager. **Required.**
commonManagerKey - An identifier that is provided to the createObject method of ObjectFactory for create CommonManager. **Required.**
invoiceStatusDAO - An identifier that is provided to the createObject method of ObjectFactory for create InvoiceStatusDAO. **Required.**
idGeneratorKey; - An identifier that is provided to the createObject method of ObjectFactory for create IDGenerator. **Required.**
expenseManagerKey - An identifier that is provided to the createObject method of ObjectFactory for create ExpenseManagaer. **Required.**
fixedBillingManagerKey - An identifier that is provided to the createObject method of ObjectFactory for create FixedBillingManager. **Required.**
dbConnectionFactoryKey - An identifier that is provided to the createObject method of ObjectFactory for create DBConnectionFactory. **Required.**
projectUtilityKey - An identifier that is provided to the createObject method of ObjectFactory for create Project Utility. **Required.**
serviceDetailManagerKey - An identifier that is provided to the createObject method of ObjectFactory for create ServiceDetailManager. **Required.**

searchBundleManagerNamespace - This is the namespace that is used to create a SearchBundleManager for retrieve the SearchBundles. **Required.**

For InvoiceManagerDelegate:

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. **Optional**
Property Name: *jndiLocation*
Property Description: This is the location name used to retrieve the home object of the respective session bean. If not specified, then the default provided by JNDIUtils is used. **Optional**

3.3Dependencies Configuration

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

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

Please see the **ejb-jar.xml** file in docs directory for the deployment descriptor for the J2EE layer of this component.

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

```
// Create an InvoiceManagerDelegate
InvoiceManager invoiceManager = new InvoiceManagerDelegate(“invoiceManagerDelegate");

// Create a new Invoice
Invoice invoice = new Invoice();
invoice.setCompanyId(7);
invoice.setDueDate(new Date());
invoice.setExpenseEntries(new ExpenseEntry[] { new ExpenseEntry() });
invoice.setFixedBillingEntries(new FixedBillingEntry[] { new FixedBillingEntry() });
invoice.setInvoiceDate(new Date());
invoice.setInvoiceNumber("1067");
// Set the status and retrieve the status from the description
invoice.setInvoiceStatus(invoiceManager.getInvoiceStatus("created"));
invoice.setPaid(false);
invoice.setPaymentTerm(new PaymentTerm());
invoice.setProjectId(198);
invoice.setPurchaseOrderNumber("1896");
invoice.setSalesTax(new BigDecimal(2000));
invoice.setServiceDetails(new ServiceDetail[] { new ServiceDetail() });

// Store in persistence with auditing
invoiceManager.addInvoice(invoice, true);

// Search the Invoices with an invoice date within a given inclusive
// date range and return only informations about Invoice object
Calendar calendar = Calendar.getInstance();
calendar.set(Calendar.MONTH, Calendar.JANUARY);
calendar.set(Calendar.DAY_OF_MONTH, 1);
calendar.set(Calendar.YEAR, 2006);
Date from = calendar.getTime();

calendar.set(Calendar.MONTH, Calendar.FEBRUARY);
calendar.set(Calendar.DAY_OF_MONTH, 15);
Date to = calendar.getTime();

// Create the filter
Filter invoiceDateFilter = InvoiceInformixFilterFactory.createInvoiceDateFilter(from, to);
Invoice[] invoices =
```

```
invoiceManager.searchInvoices(invoiceDateFilter,InvoiceSearchDepth.INVOICE_ONLY);

//Look if is possible to create Invoice
boolean canCreateInvoice=invoiceManager.canCreateInvoice(210);

//Enumerate all invoices
invoices=invoiceManager.getAllInvoices();

//Retrieve an invoice by id
invoice=invoiceManager.getInvoice(800);

//Update an existing invoice
invoiceManager.updateInvoice(invoice, true);

//Enumerate all statuses
InvoiceStatus[]invoiceStatuses=invoiceManager.getAllInvoicesStatus();

//Retrieve an InvoiceStatus by id
InvoiceStatus invoiceStatus=invoiceManager.getInvoiceStatus(80);

//use the NOT,AND and OR filters for gruop several filters and search with them
Filter filter=new
AndFilter(InvoiceInformixFilterFactory.createClientIdFilter(1),InvoiceInformixFilterFactory.createC
ompanyIdFilter(7));
invoices=invoiceManager.searchInvoices(filter, InvoiceSearchDepth.INVOICE_ONLY);

filter=new
OrFilter(InvoiceInformixFilterFactory.createClientIdFilter(1),InvoiceInformixFilterFactory.createCo
mpanyIdFilter(7));
invoices=invoiceManager.searchInvoices(filter, InvoiceSearchDepth.INVOICE_ONLY);

filter=new NotFilter(InvoiceInformixFilterFactory.createCompanyIdFilter(7));
invoices=invoiceManager.searchInvoices(filter, InvoiceSearchDepth.INVOICE_ONLY);
```

5.Future Enhancements

Provide implementations for different RDBMS.