Auction Persistence Requirements Specification

1. Scope

1.1 Overview

The Auction Persistence component provides the Orpheus application with an interface to persistent storage of auction data. The central persistence functionality is handled by a stateless session EJB, but for interoperation with the Auction Framework component the bean is wrapped in an ordinary class.

1.2 Logic Requirements

1.2.1 Stateless Session Bean for Auction Data Persistence

The component will provide a stateless session EJB through which it will access auction data stored in an SQL Server database. The component will not rely on collocation of the client and EJB container. It is not required to use entity beans, and if it does use them it will not expose them to component clients.

1.2.2 Auction Framework Persistence Plug-in

The component will provide a `CustomAuctionPersistence` class, an implementation of the Auction Framework's `AuctionPersistence` interface that operates as a client of the persistence EJB (section 1.2.1) to adapt it to the Auction Framework API.

1.2.3 Custom Bid Implementation

The component will provide a custom `Bid` implementation with which to transport an image ID and unique `Bid` ID along with the other bid information. The component will produce no `Bids` other than instances of this `Bid` implementation, and is permitted to assume that `Bid` objects provided to it are also instances of this class. Here is a partial representation of the required class:

```
package com.orpheus.auction.persistence;

/**
 * A Bid implementation that carries a unique ID and image ID in
 * addition to standard Bid data
 */
public class CustomBid implements com.topcoder.util.auction.Bid {

      /**
       * Creates a new CustomBid with the specified characteristics.
       * The bid's unique ID is initially null
       */
      public CustomBid(long bidderId, long imageId, int maxAmount,
            Date timestamp) {
      }

      /**
       * Retrieves this bid's unique ID, which may be null if none has
       * yet been assigned
       */
      public Long getId() {
      }
```

```
            /**
             * Sets this bid’s unique ID to the specified value
             */
            void setId(long id) {
            }

            /**
             * Retrieves this bid’s associated image ID
             */
            public long getImageId() {
            }
      }
```

1.2.4 Caching

The Auction Framework persistence plug-in and associated bean will employ a caching strategy that reduces the number of read requests made across layers (plugin to (remote) bean and bean to DBMS). Writes will not be delayed, but their results may be cached.

1.2.5 Thread Safety

The component will be used in a multithreaded application server environment, and therefore must be thread safe.

1.2.6 Local and Remote Interfaces

The EJB provided by this component will provide both local and remote interfaces.

1.2.7 Deployment Descriptors

The component deliverables will include a sample deployment descriptor for the EJB.

1.3 Required Algorithms

None

1.4 Example of the Software Usage

The component will be used to provide access to auction data for the Orpheus application.

1.5 Future Component Direction

None planned.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None

2.1.2 External Interfaces

2.1.2.1 Generic Component Interfaces

Refer to the Auction Framework specifications for the `AuctionPersistence` interface.

2.1.2.2 Database Schema

The component will interact with a database partially represented by the following SQL script. The most important tables for this component are the auction, bid, and effective_bid tables; these are the only ones updated by the component. Partial descriptions of several other tables are presented as well, enough to elucidate all the relevant relstionships.

```
-- -------------------------------------------------------------
-- Any_user entities represent the identifying and authorization
-- information for specific users.
-- -------------------------------------------------------------
CREATE TABLE any_user (
  id BIGINT NOT NULL,
  handle VARCHAR(29) NOT NULL,
  e_mail VARCHAR(255) NOT NULL,
  PRIMARY KEY(id),
  UNIQUE INDEX user_handle_unique(handle),
  UNIQUE INDEX user_email_unique(e_mail)
)

-- -------------------------------------------------------------
-- Game entities represent individual games managed by the application.
-- -------------------------------------------------------------
CREATE TABLE game (
  id BIGINT NOT NULL AUTO_INCREMENT,
  PRIMARY KEY(id),
)

-- -------------------------------------------------------------
-- Sponsor entities represent game sponsor representatives
-- eligible to place bids.
-- -------------------------------------------------------------
CREATE TABLE sponsor (
  any_user_id BIGINT NOT NULL,
  PRIMARY KEY(any_user_id),
  FOREIGN KEY(any_user_id)
    REFERENCES any_user(id)
)

-- -------------------------------------------------------------
-- Domain entities represent web sites that (it is hoped) are
-- eligible to host the application.  Each is associated with a
-- specific sponsor
-- -------------------------------------------------------------
CREATE TABLE domain (
  id BIGINT NOT NULL AUTO_INCREMENT,
  sponsor_id BIGINT NOT NULL,
  PRIMARY KEY(id),
  FOREIGN KEY(sponsor_id)
    REFERENCES sponsor(any_user_id)
)
```

```
-- ------------------------------------------------------------
-- Image entities represent the domain-specific images associated
-- with sponsored domains.
-- ------------------------------------------------------------
CREATE TABLE image (
  id BIGINT NOT NULL AUTO_INCREMENT,
  domain_id BIGINT NOT NULL,
  PRIMARY KEY(id),
  FOREIGN KEY(domain_id)
    REFERENCES domain(id)
)

-- ------------------------------------------------------------
-- Hosting_block entities aggregate hosting slots, providing for
-- group-wise sequencing and provision of slots.
-- ------------------------------------------------------------
CREATE TABLE hosting_block (
  id BIGINT NOT NULL AUTO_INCREMENT,
  PRIMARY KEY(id)
)

-- ------------------------------------------------------------
-- Represents an auction for slots from a hosting block.  Each auction
-- has a specific start time and end time, a minimum bid, a minimum bid
-- increment, and a count of individual items for sale.  Slots are
-- auctioned anonymously, so that the n highest bids in each auction
-- win slots (where n is the number of slots in the block).
-- ------------------------------------------------------------
CREATE TABLE auction (
  hosting_block_id BIGINT NOT NULL,
  start_time DATETIME NOT NULL,
  end_time DATETIME NOT NULL,
  min_bid INTEGER UNSIGNED NOT NULL,
  bid_increment INTEGER UNSIGNED NOT NULL,
  item_count INTEGER UNSIGNED NOT NULL,
  PRIMARY KEY(hosting_block_id),
  FOREIGN KEY(hosting_block_id)
    REFERENCES hosting_block(id)
)

-- ------------------------------------------------------------
-- Represents a bid placed by a sponsor to host the application at a
-- specific domain using a specific domain image.  (Domain is implied
-- by image, and sponsor by domain.) The auto-incrementing id of this
-- table serves to establish a total order of bids (reception order),
-- even when two or more are received at the same time within the
-- granularity of the timestamp.  The sponsor also specifies the
-- maximum amount he is willing to pay, for the purpose of automated
-- proxy bidding.  The actual current amount that the sponsor would pay
-- if this bid won (often less than max_amount) is specified
-- separately, via an effective_bid entity.
-- ------------------------------------------------------------
CREATE TABLE bid (
  id BIGINT NOT NULL AUTO_INCREMENT,
  auction_id BIGINT NOT NULL,
  image_id BIGINT NOT NULL,
  max_amount INTEGER UNSIGNED NOT NULL,
  time DATETIME NOT NULL,
  PRIMARY KEY(id),
  FOREIGN KEY(auction_id)
    REFERENCES auction(hosting_block_id)
  FOREIGN KEY(image_id)
    REFERENCES image(id)
)

-- -------------------------------------------------------------
-- An effective_bid entity represents the amount that would be paid by
-- the bidding sponsor if the associated bid won.  This may be updated
-- periodically by the auction engine in response to automated proxy
-- bidding on behalf of the bidding sponsor.  A bid that has been
-- outbid will not have an associated effective_bid.
-- -------------------------------------------------------------
CREATE TABLE effective_bid (
  bid_id BIGINT NOT NULL,
  current_amount INTEGER UNSIGNED NOT NULL,
  PRIMARY KEY(bid_id),
  FOREIGN KEY(bid_id)
    REFERENCES bid(id)
)
```

2.1.2.3 Property Correspondence

The correspondences between object properties and database fields is described by this table:

Object property	**Database field**
Auction.id	auction.hosting_block_id
Auction.summary	(none)*
Auction.description	(none)†
Auction.itemCount	auction.item_count
Auction.minimumBid	auction.min_bid
Auction.startDate	auction.start_time
Auction.endDate	auction.end_time
Bid.bidderId	sponsor.any_user_id‡
Bid.effectiveAmount	effective_bid.current_amount
Bid.maxAmount	bid.max_amount
Bid.timestamp	bid.time
CustomBid.id	bid.id
CustomBid.imageId	bid.image_id

* Any summary submitted by the application is ignore on auction creation; an empty string is provided on auction retrieval

† Any description submitted by the application is ignore on auction creation; an empty string is provided on auction retrieval

‡ On creation or update, the provided bidder ID is checked against the sponsor ID (spondor.any_user_id) associated with the domain to which the specified image ID is assigned, but the bidder ID is not actually stored. On retrieval, the bidder ID is extracted based on the image ID.

2.1.3 Environment Requirements

- Development language: Java1.4

- Compile target: Java1.4
- J2EE 1.4

2.1.4 Package Structure

com.orpheus.auction.persistence

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

None

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

EJB 2.1

SQL 92

3.2.2 TopCoder Software Component Dependencies:

Auction Framework 1.0

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

3.2.3 Third Party Component, Library, or Product Dependencies:

None

3.2.4 QA Environment:

- RedHat Enterprise Linux 4
- JBoss Application Server 4.0.4
- Microsoft SQL Server 2005

3.3 Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines. Modifications to these guidelines for this component should be detailed below.

3.4 Required Documentation

3.4.1 Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification
- Sample Deployment Descriptors

3.4.2 Help / User Documentation

- Design documents must clearly define intended component usage in the ‘Documentation’ tab of Poseidon.