Administration Persistence Requirements Specification

1. Scope

1.1 Overview

The Administration Persistence component provides the Orpheus application with an interface to persistent storage of administrative data. The central persistence functionality is handled by a stateless session EJB.

1.2 Logic Requirements

1.2.1 Stateless Session Bean for Administrative Data Persistence

The component will provide a stateless session EJB by which persistent administrative data can be retrieved and manipulated. The EJB is not required to use entity beans; if it does use them then it must not expose instances to its clients. The bean will provide the following services via the specified home and component interfaces:

1.2.1.1 Retrieve Admin Summary Report

On request, the bean will prepare a summary object suggesting administrative tasks that are pending. Specifically, the summary will report

- the number of active games that have winners pending,
- the number of sponsors awaiting approval, and
- the total number of active games. (Those started but not yet won by anyone.)

1.2.1.2 Retrieve Pending Winners

On request, the bean will provide a report of the pending game winners, in the form of an array of objects indicating

- the ID of the player who is a pending winner,
- the ID of the game they have provisionally won, and
- the payout to be made to the winner in the event that the win is approved

The objects will be ordered in the array in the sequence in which the provisional wins occurred, and they may include more than one provisional winner for the same game. They are obtained by examination of the 'plyr_compltd_game' table. Payout is computed as follows:

- all hosting slots that were used in the specified game (those that ever started hosting) are identified
- the sum of the effective bid amounts for all the identified slots is computed
- the sum is reduced by an administration fee computed as a fraction of the total; the specific fraction used is to be determined from component configuration

Note that hosting slots' association with games is indirect, from slot to bid to auction to hosting block to game.

1.2.1.3 Approve / Reject Pending Winner

The component will provide the ability to approve or reject a pending winner. For each pending winner, the client is expected to issue at most one approval or rejection. For each game with a pending winner, the client is expected to issue at most one approval.

When a winner is rejected, the 'plyr_compltd_game' table is updated to set the 'is_handled' field of the corresponding row to true.

When a winner is approved,

- All rows of the 'plyr_compltd_game' table that reference the corresponding game are updated by setting their 'is_handled' fields to true
- A row is inserted into the 'plyr_won_game' table indicating the player, game, payout, and approval timestamp

1.2.1.4 Approve / Reject Domain

The component will provide the ability to set the approval flag on sponsors' domains, identified by their unique IDs.

1.2.1.5 Approve / Reject Image

The component will provide the ability to set the approval flag on sponsors' domain images, identified by their unique IDs.

1.2.1.6 Record Puzzles and Brain Teasers

The component will provide the ability to record puzzle data in the application's database, supporting game-win puzzles and hosting slot brain teasers alike. Puzzle data will be supplied in the form of serializable objects implementing the `PuzzleData` interface defined by the Puzzle Framework component.

1.2.1.7 Internal Interfaces

The component will provide and use the following interfaces:

```
package com.orpheus.administration.persistence;

/**
 * The (remote) home interface of the Admin Data EJB
 */
public interface AdminDataHome extends javax.ejb.EJBHome {

      /**
       * Creates a new AdminData object and returns it.
       *
       * @return
       */
      public AdminData create() throws java.rmi.RemoteException,
            javax.ejb.CreateException;
}

/**
 * The local home interface of the Admin Data EJB
 */
public interface AdminDataLocalHome extends javax.ejb.EJBLocalHome {

      /**
       * Creates a new AdminDataLocal object and returns it.
       *
       * @return
       */
```

```
        public AdminDataLocal create() throws javax.ejb.CreateException;
}

/**
 * The (remote) component interface for the AdminData EJB, which is
 * used to record and access application data particular to
 * administrative functions.
 */
public interface AdminData extends javax.ejb.EJBObject {

        /**
         * Retrieves an administrative data summary from the persistent
         * store
         *
         * @return
         * @throws RemoteException if a communication error occurs
         *    between the client and EJB container
         * @throws AdministrationException if a checked exception
         *    occurring within the bean implementation prevents this
         *    method from completing successfully
         */
        public AdminSummary getAdminSummary()
              throws java.rmi.RemoteException,
              com.orpheus.administration.AdministrationException;

        /**
         * Sets the approval flag for the specified domain
         *
         * @param domainId
         * @param approved
         * @throws RemoteException if a communication error occurs
         *    between the client and EJB container
         * @throws AdministrationException if a checked exception
         *    occurring within the bean implementation prevents this
         *    method from completing successfully
         */
        public void setDomainApproval(long domainId, boolean approved)
              throws java.rmi.RemoteException,
              com.orpheus.administration.AdministrationException;

        /**
         * Sets the approval flag for the specified image
         *
         * @param imageId
         * @param approved
         * @throws RemoteException if a communication error occurs
         *    between the client and EJB container
         * @throws AdministrationException if a checked exception
         *    occurring within the bean implementation prevents this
         *    method from completing successfully
         */
        public void setImageApproval(long imageId, boolean approved)
              throws java.rmi.RemoteException,
              com.orpheus.administration.AdministrationException;
```

```
/**
 * Records the specified PuzzleData objects in the application's
 * database, returning an array of the unique IDs assigned to
 * them (in the same order as the puzzles appear in the argument)
 *
 * @param puzzles
 * @return
 * @throws RemoteException if a communication error occurs
 *    between the client and EJB container
 * @throws AdministrationException if a checked exception
 *    occurring within the bean implementation prevents this
 *    method from completing successfully
 */
public long[] storePuzzles(
      com.topcoder.util.puzzle.PuzzleData[] puzzles)
      throws java.rmi.RemoteException,
      com.orpheus.administration.AdministrationException;

/**
 * Retrieves the currently-pending winners' information
 *
 * @return
 * @throws RemoteException if a communication error occurs
 *    between the client and EJB container
 * @throws AdministrationException if a checked exception
 *    occurring within the bean implementation prevents this
 *    method from completing successfully
 */
public PendingWinner[] getPendingWinners()
      throws java.rmi.RemoteException,
      com.orpheus.administration.AdministrationException;

/**
 * Approves the specified PendingWinner's win
 *
 * @param winner
 * @param date
 * @throws RemoteException if a communication error occurs
 *    between the client and EJB container
 * @throws AdministrationException if a checked exception
 *    occurring within the bean implementation prevents this
 *    method from completing successfully
 */
public void approveWinner(PendingWinner winner,
      java.util.Date date) throws java.rmi.RemoteException,
      com.orpheus.administration.AdministrationException;

/**
 * Rejects the specified PendingWinner's win
 *
 * @param winner
 * @throws RemoteException if a communication error occurs
 *    between the client and EJB container
```

```
         * @throws AdministrationException if a checked exception
         *    occurring within the bean implementation prevents this
         *    method from completing successfully
         */
        public void rejectWinner(PendingWinner winner)
              throws java.rmi.RemoteException,
              com.orpheus.administration.AdministrationException;
}

/**
 * The local component interface for the AdminData EJB, which is used
 * to record and access application data particular to administrative
 * functions.
 */
public interface AdminDataLocal extends javax.ejb.EJBLocalObject {

      /**
       * Retrieves an administrative data summary from the persistent
       * store
       *
       * @return
       * @throws AdministrationException if a checked exception
       *    occurring within the bean implementation prevents this
       *    method from completing successfully
       */
      public AdminSummary getAdminSummary()
            throws com.orpheus.administration.AdministrationException;

      /**
       * Sets the approval flag for the specified domain
       *
       * @param domainId
       * @param approved
       * @throws AdministrationException if a checked exception
       *    occurring within the bean implementation prevents this
       *    method from completing successfully
       */
      public void setDomainApproval(long domainId, boolean approved)
            throws com.orpheus.administration.AdministrationException;

      /**
       * Sets the approval flag for the specified image
       *
       * @param imageId
       * @param approved
       * @throws AdministrationException if a checked exception
       *    occurring within the bean implementation prevents this
       *    method from completing successfully
       */
      public void setImageApproval(long imageId, boolean approved)
            throws com.orpheus.administration.AdministrationException;

      /**
       * Records the specified PuzzleData objects in the application's
```

```
         * database, returning an array of the unique IDs assigned to
         * them (in the same order as the puzzles appear in the argument)
         *
         * @param puzzles
         * @return
         * @throws AdministrationException if a checked exception
         *    occurring within the bean implementation prevents this
         *    method from completing successfully
         */
        public long[] storePuzzles(
              com.topcoder.util.puzzle.PuzzleData[] puzzles)
              throws com.orpheus.administration.AdministrationException;

        /**
         * Retrieves the currently-pending winners' information
         *
         * @return
         * @throws AdministrationException if a checked exception
         *    occurring within the bean implementation prevents this
         *    method from completing successfully
         */
        public PendingWinner[] getPendingWinners()
              throws com.orpheus.administration.AdministrationException;

        /**
         * Approves the specified PendingWinner's win
         *
         * @param winner
         * @param date
         * @throws AdministrationException if a checked exception
         *    occurring within the bean implementation prevents this
         *    method from completing successfully
         */
        public void approveWinner(PendingWinner winner,
              java.util.Date date)
              throws com.orpheus.administration.AdministrationException;

        /**
         * Rejects the specified PendingWinner's win
         *
         * @param winner
         * @throws AdministrationException if a checked exception
         *    occurring within the bean implementation prevents this
         *    method from completing successfully
         */
        public void rejectWinner(PendingWinner winner)
              throws com.orpheus.administration.AdministrationException;
    }

    /**
     * An interface representing summary administrative information
     */
    public interface AdminSummary extends java.io.Serializable {
```

```
        /**
         * Returns the number of sponsors that are pending admin approval
         *
         * @return
         */
        public int getPendingSponsorCount();

        /**
         * Returns the number of games in which a win is pending
         *
         * @return
         */
        public int getPendingWinnerCount();

        /**
         * Returns the number of active games
         *
         * @return
         */
        public int getActiveGameCount();
}

/**
 * Represents the essential details associating a pending game winner
 * with the game he has provisionally won and the payout he will
 * receive if approved
 */
public interface PendingWinner extends java.io.Serializable {

        /**
         * Records the specified PuzzleData objects in the application's
         * database, returning an array of the unique IDs assigned to
         * them (in the same order as the puzzles appear in the argument)
         *
         * @return
         */
        public long getPlayerId();

        /**
         * Returns the ID of the game the player has provisionally won
         *
         * @return
         */
        public long getGameId();

        /**
         * Returns the payout the player stands to receive
         *
         * @return
         */
        public int getPayout();
}
```

1.2.2 *Game Messaging Persistence*

The component will provide a `MessengerPlugin` (see Messenger Framework 1.0 documentation) and a complementary `DataStore` (see RSS Generator 2.0) for making general announcements by recording the relevant data in the application's database and subsequently returning it. These will be based on a stateless session EJB (to be provided as well) that mediates the data transfers. These will consume and repackage (respectively) the following data:

- A URI to use as a globally unique identifier
- A category string
- A content type string
- A message body
- A timestamp

1.2.3 *Caching*

The provided EJBs will employ a caching strategy that reduces the number of read requests made across layers (bean to DBMS). Writes will not be delayed, but their results may be cached.

1.2.4 *Thread Safety*

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

1.2.5 *Deployment Descriptors*

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

1.2.6 *Database*

The component will interact with a database partially defined by this SQL script:

```
-- ------------------------------------------------------------
-- any_user entities represent the identifying and authorization
-- information for specific players, admins, and sponsors.
-- ------------------------------------------------------------

CREATE TABLE any_user (
  id BIGINT NOT NULL,
  PRIMARY KEY(id),
)

-- ------------------------------------------------------------
-- users who are sponsors
-- ------------------------------------------------------------

CREATE TABLE sponsor (
  any_user_id BIGINT NOT NULL,
  is_approved BOOL NULL,
  PRIMARY KEY(any_user_id),
  FOREIGN KEY(any_user_id)
    REFERENCES any_user(id)
)

-- ------------------------------------------------------------
```

```
-- Users who are players
-- -------------------------------------------------------------

CREATE TABLE player (
  any_user_id BIGINT NOT NULL,
  PRIMARY KEY(any_user_id),
  FOREIGN KEY(any_user_id)
    REFERENCES any_user(id)
)

-- -------------------------------------------------------------
-- Download_obj entities represent binary objects of specified media
-- type. They are generally intended to be provided for download from
-- the server, but can be used internally instead.
-- -------------------------------------------------------------

CREATE TABLE download_obj (
  id BIGINT NOT NULL AUTO_INCREMENT,
  media_type VARCHAR(255) NOT NULL,
  content BLOB NOT NULL,
  PRIMARY KEY(id)
)

-- -------------------------------------------------------------
-- A puzzle entity is a generic representation of a puzzle of a puzzle:
-- a brain-teaser, word puzzle, tile puzzle, jigsaw puzzle, etc.  It
-- provides a reference with which to associate any number of
-- "attributes" and "resources".
-- -------------------------------------------------------------

CREATE TABLE puzzle (
  id BIGINT NOT NULL AUTO_INCREMENT,
  name VARCHAR(255) NULL,
  PRIMARY KEY(id)
)

-- -------------------------------------------------------------
-- Puzzle_attribute entities associate textual attributes of coded
-- types with specific puzzles.
-- -------------------------------------------------------------

CREATE TABLE puzzle_attribute (
  id BIGINT NOT NULL AUTO_INCREMENT,
  puzzle_id BIGINT NOT NULL,
  name VARCHAR(255) NOT NULL,
  value VARCHAR(255) NOT NULL,
  PRIMARY KEY(id),
  UNIQUE INDEX attribute_names_unique(puzzle_id, name),
  FOREIGN KEY(puzzle_id)
    REFERENCES puzzle(id)
)
-- -------------------------------------------------------------
-- Puzzle_resource entities associate arbitrary binary objects with
-- specific puzzle entities.  They provide coded resource type
```

```
-- information, textual resource information.
-- ------------------------------------------------------------

CREATE TABLE puzzle_resource (
  id BIGINT NOT NULL AUTO_INCREMENT,
  puzzle_id BIGINT NOT NULL,
  name VARCHAR(255) NOT NULL,
  download_obj_id BIGINT NOT NULL,
  PRIMARY KEY(id),
  UNIQUE INDEX resource_name_unique(puzzle_id, name),
  FOREIGN KEY(puzzle_id)
    REFERENCES puzzle(id)
  FOREIGN KEY(download_obj_id)
    REFERENCES download_obj(id)
)

-- ------------------------------------------------------------
-- Game entities represent individual games provided by the
-- application.
-- ------------------------------------------------------------

CREATE TABLE game (
  id BIGINT NOT NULL AUTO_INCREMENT,
  start_date DATETIME NOT NULL,
  PRIMARY KEY(id),
)

-- ------------------------------------------------------------
-- The plyr_regstrd_game relation establishes which players
-- are registered for which games.
-- ------------------------------------------------------------

CREATE TABLE plyr_regstrd_game (
  game_id BIGINT NOT NULL,
  player_id BIGINT NOT NULL,
  PRIMARY KEY(game_id, player_id),
  FOREIGN KEY(player_id)
    REFERENCES player(any_user_id)
  FOREIGN KEY(game_id)
    REFERENCES game(id)
)

-- ------------------------------------------------------------
-- The plyr_compltd_game relation establishes which players have
-- completed which games, and provides a global serial number by
-- which to determine the order in which the players achieved their
-- game completions.
-- ------------------------------------------------------------

CREATE TABLE plyr_compltd_game (
  game_id BIGINT NOT NULL,
  player_id BIGINT NOT NULL,
  sequence_number BIGINT NOT NULL AUTO_INCREMENT,
  is_handled BOOL NOT NULL DEFAULT false,
```

```
  PRIMARY KEY(game_id, player_id),
  FOREIGN KEY(game_id)
    REFERENCES game(id)
  FOREIGN KEY(player_id)
    REFERENCES player(any_user_id)
)

-- ---------------------------------------------------------
-- Domain entities represent web sites that (it is hoped) are
-- eligible to be host sites.  Each is associated with a specific
-- sponsor, and is characterized by the base URL of the site.
-- ---------------------------------------------------------

CREATE TABLE domain (
  id BIGINT NOT NULL AUTO_INCREMENT,
  sponsor_id BIGINT NOT NULL,
  base_url VARCHAR(255) NOT NULL,
  is_approved BOOL NULL,
  PRIMARY KEY(id),
  UNIQUE INDEX dmn_spn_url_unique(base_url),
  FOREIGN KEY(sponsor_id)
    REFERENCES sponsor(any_user_id)
)

-- ---------------------------------------------------------
-- Records game-win events, identifying the winner, win approval
-- timestamp, and payout amount.  Presence of a row in this table
-- for a specific game ID signals that that game is over.
-- ---------------------------------------------------------

CREATE TABLE plyr_won_game (
  game_id BIGINT NOT NULL,
  player_id BIGINT NOT NULL,
  date DATETIME NOT NULL,
  payout INTEGER UNSIGNED NOT NULL,
  PRIMARY KEY(game_id),
  FOREIGN KEY(player_id)
    REFERENCES player(any_user_id)
  FOREIGN KEY(game_id)
    REFERENCES game(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,
  game_id BIGINT NOT NULL,
  PRIMARY KEY(id)
  FOREIGN KEY(game_id)
    REFERENCES game(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)
)

-- ------------------------------------------------------------
-- Image entities represent the domain-specific images associated
-- with sponsored domains.  In addition to a specific domain, each is
-- associated with a download_obj containing the binary image data.
-- Each bears a flag indicating whether the image has been approved or
-- rejected, if any such decision has yet been made.
-- ------------------------------------------------------------

CREATE TABLE image (
  id BIGINT NOT NULL AUTO_INCREMENT,
  domain_id BIGINT NOT NULL,
  download_obj_id BIGINT NOT NULL,
  is_approved BOOL NULL,
  description VARCHAR(255) NOT NULL,
  PRIMARY KEY(id),
  FOREIGN KEY(domain_id)
    REFERENCES domain(id)
  FOREIGN KEY(download_obj_id)
    REFERENCES download_obj(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 actual
-- current amount that the sponsor would pay if this bid won is
-- specified separately, via an effective_bid entity.
-- ------------------------------------------------------------

CREATE TABLE bid (
  id BIGINT NOT NULL AUTO_INCREMENT,
  image_id BIGINT NOT NULL,
  PRIMARY KEY(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.
-- ---------------------------------------------------------

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)
)

-- ---------------------------------------------------------
-- Hosting_slot entities represent specific application appearances
-- on particular domains.  They are associated with particular hosting
-- blocks indirectly through bids, and through blocks with specific
-- games.  Each has an associated collection of related brain-teasers,
-- a chain of target objects to find, key details, and a game-win
-- puzzle.  Each is ordered relative to the other slots in its block by
-- a sequence number, and has associated times when it started and
-- stopped hosting the Ball, along with a limiting number of minutes
-- that the Ball may remain in this slot.
-- ---------------------------------------------------------

CREATE TABLE hosting_slot (
  id BIGINT NOT NULL AUTO_INCREMENT,
  bid_id BIGINT NOT NULL,
  sequence_number INTEGER UNSIGNED NOT NULL,
  hosting_start DATETIME NULL,
  hosting_end DATETIME NULL,
  PRIMARY KEY(id),
  FOREIGN KEY(bid_id)
    REFERENCES bid(id)
)

-- ---------------------------------------------------------
-- The plyr_compltd_slot relation establishes which players have
-- 'completed' (i.e. found a key or the Ball associated with)  which
-- hosting slots, and records a timestamp of when they did so.
-- ---------------------------------------------------------

CREATE TABLE plyr_compltd_slot (
  hosting_slot_id BIGINT NOT NULL,
  player_id BIGINT NOT NULL,
  timestamp DATETIME NOT NULL,
  key_text VARCHAR(8) NOT NULL,
  key_image_id BIGINT NOT NULL,
  PRIMARY KEY(hosting_slot_id, player_id),
  FOREIGN KEY(player_id)
```

```
    REFERENCES player(any_user_id)
  FOREIGN KEY(hosting_slot_id)
    REFERENCES hosting_slot(id)
  FOREIGN KEY(key_image_id)
    REFERENCES download_obj(id)
)

-- ------------------------------------------------------------
-- Represents a generic message with category, update timestamp,
-- and typed content.
-- ------------------------------------------------------------

CREATE TABLE message (
  id BIGINT NOT NULL AUTO_INCREMENT,
  guid VARCHAR(255) NOT NULL,
  category VARCHAR(20) NOT NULL,
  content_type VARCHAR(255) NOT NULL,
  update_time DATETIME NOT NULL,
  content TEXT NOT NULL,
  PRIMARY KEY(id),
  UNIQUE INDEX guid_uique(guid)
)
```

1.3 Required Algorithms

None

1.4 Example of the Software Usage

The component will be used to provide access to administrative 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

Refer to components' documentation.

2.1.3 Environment Requirements

- Development language: Java1.4
- Compile target: Java1.4
- J2EE 1.4

2.1.4 Package Structure

com.orpheus.administration.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:

Messenger Framework 1.0

Puzzle Framework 1.0

Orpheus Administration Logic 1.0

RSS Generator 2.0

**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.