Game Persistence Requirements Specification

1. Scope

1.1 Overview

The Orpheus Game Persistence component provides the Orpheus application with an interface to persistent storage of game 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 Game Data Persistence

The component will provide a stateless session EJB through which it will access game data stored in an SQL Server database. The component will not rely on collocation of the client and EJB container, though it will provide both local and remote interfaces. It is not required to use entity beans, and if it does use them it will not expose them to component clients. The bean will provide the following behaviors:

1.2.1.1 Create a New Game

The bean will provide a method that creates a new game record in the database, based on user-provided details. The method will accept a `Game` object from which it will use the following information:

- *ID*, if not null. If the ID is null then the component will allow the database to automatically generate an ID (this is expected to be the usual case). The automatically-generated IDs will be small positive integers.
- *Ball color.* (The ID is used, and must match an existing ball_color entity; no new ball_color is created.)
- *Required Key Count*.
- *Start Date*.
- *Hosting Blocks*. For each `HostingBlock` object among the provided game data, the bean will create a hosting block in the same manner described in section 1.2.1.4.

Other game parameters are not set at this point, and in particular, no hosting slots are created for the game by this method. The method will return a `Game` object representing the actual persistent state of the newly-created game.

1.2.1.2 Create Hosting Slots for an Existing Game

The bean will provide a method that creates new hosting slots in an existing hosting block based on the block ID and the IDs of the bids on which the slots will be based. The bean verifies that all specified bid IDs refer to bids in the auction associated with the specified hosting block, then proceeds to create the hosting_slot entities in the database. The new slots will be assigned ascending, positive, consecutive sequence numbers that are as small as possible without duplicating the sequence numbers of any hosting slots already associated with the block. Slot IDs will be automatically generated, and hosting timestamps will initially be null. The method will return an array of `HostingSlot` objects representing the new slots.

1.2.1.3 Create a New Domain Record

The bean will provide a method that creates a new domain record and associated image records based on a `Domain` object and associated `ImageInfo` objects provided by the user. Where `Domain` or `ImageInfo` IDs are null, the bean will generate suitable IDs automatically; all other fields of the domain and image records will be copied from the provided objects. The method will return a `Domain` object with associated `ImageInfo` objects that reflect the actual database records created.

1.2.1.4 Add a Hosting Block for a Game

The bean will provide a method that adds a new hosting_block record to the database, associated with a specified game. The method will accept the game ID and the maximum hosting time for the block's slots as parameters, and it will assign the minimum positive sequence number greater than any other sequence numbers for the specified game's blocks. The new block will not initially have any associated hosting slots (and no auction). The method will return a `HostingBlock` object describing the new block.

1.2.1.5 Retrieve Game Details

The bean will provide a method for obtaining full information about a particular game, specified by its ID. It will return a `Game` object containing the available information about the specified game, its hosting blocks, and their hosting slots.

1.2.1.6 Retrieve Hosting Block Details

The bean will provide a method for obtaining full information about a particular hosting block, specified by its ID. It will return a `HostingBlock` object containing the available information about the specified block and its associated hosting slots.

1.2.1.7 Retrieve Hosting Slot Details

The bean will provide a method for obtaining full information about a particular hosting slot, specified by its ID. It will return a `HostingSlot` object containing the available information about the specified slot.

1.2.1.8 Retrieve Downloadable Data

The bean will provide a method to retrieve downloadable data from the database. It will accept the download ID as a parameter, and will return a `DownloadData` object representing the data and its associated metadata. (Note that the `DownloadData` implementation used will need to be serializable.)

1.2.1.9 Retrieve Domain Details

The bean will provide a method to retrieve domain information from the database. It will accept the domain ID as a parameter and will return a `Domain` object containing the domain details.

1.2.1.10 Retrieve Keys for Player

The bean will provide a method for retrieving the keys associated with a specified list of slots, for a specified player. The player will be specified by player ID and the slots by an array of slot IDs, and the method will return an array of the corresponding key text strings.

1.2.1.11 Retrieve Puzzle Data

The bean will provide a method by which to retrieve puzzle data from the database. The method will accept the puzzle data ID as a parameter, and will return a corresponding `PuzzleData` object.

1.2.1.12 Record Download

The bean will provide a method for recording download counts in the database. It will accept the name of the download (a `String`) as a parameter, and will increment by one the download count associated with the named download. If no row yet exists for the specified download name then the method will insert a new one.

1.2.1.13 Record Registration

The bean will provide a method by which to record players' registrations for specific games. It will accept the player and game IDs as parameters and will insert a corresponding row into the `plyr_regstrd_game` table.

1.2.1.14 Record Hosting Slot Completion

The bean will provide a method by which to record players' completion of individual hosting slots and generate a suitable player-specific key. It will accept the player ID, slot ID, and completion `Date` as parameters, and perform these steps:

- Generate fixed-length key text and a corresponding image with use of the Random String Image component. Key length and alphabet will be configurable via configuration file;
- Store the image data in the `download_obj` table, with appropriate media type, as also described in requirement 1.2.1.16
- Record the data as a new row in the `plyr_compltd_slot` table.
- Compose and return a `SlotCompletion` object describing the result.

1.2.1.15 Record Game Completion

The bean will provide a method for recording a player's completion of a particular game. It will accept a player ID and game ID as parameters, and will insert a corresponding row into the `plyr_compltd_game` table.

1.2.1.16 Record Generic Object

The bean will provide a method for recording generic data in the `download_obj` table. It will accept an object name (to be provided to clients as a file name suggestion), a MIME media type string, and the object data as a byte array. It will insert these into the download_obj table as a new row and will return the ID assigned.

1.2.1.17 Update Hosting Slots

The bean will provide a method for updating hosting slot data. It will accept an array of `HostingSlot` objects to store, and these may have associated `DomainTargets`. The slots are expected to already exist in the database, but not necessarily the domain targets. The bean will assume that `DomainTarget` objects will non-null IDs correspond to existing targets (that do not need to be modified), and those with null IDs correspond to new targets to be inserted. In addition to inserting any new domain targets, the component will update the following slot properties (and no others) for each provided slot:

- sequence number
- hosting start date
- hosting end date
- puzzle ID
- brain-teaser IDs

The method will return an array of `HostingSlot` objects representing the resulting state of all the specified slots.

1.2.1.18 Delete Hosting Slot

The bean will provide a method that deletes a specified hosting slot from the database, along with associated rows from the `plyr_compltd_slot` table, and rows from the download_obj table associated with the deleted slot completions through their image IDs. (See 1.2.1.14 and 1.2.3.)

1.2.1.19 Look Up Active Domains

The bean will provide a method to look up 'active' domains. A domain is active if it is associated with a hosting slot that has hosted its game already or is currently hosting it, and that game that has not yet ended (by being won).

1.2.1.20 Look Up Games by Domain

The bean will provide a method to look up games by hosting domain, in order to determine whether particular domains should be searched by specific players. It accepts the domain name string and a player ID, and returns an array of `Game` objects representing the games that meet these criteria:

- They have already started and not yet ended
- Among those slots not already completed by the specified player, the game contains at least one associated with a domain that matches the specified name
- Among the game's slots identified by the previous criterion, at least one has started hosting the game (whether or not it has subsequently stopped).

If no games meet those criteria then the returned array has length zero.

1.2.1.21 Look Up Completed Hosting Slots

The bean will provide a method to look up all hosting slots in a particular game that have been completed by any player. It will take the game ID as a parameter, and will return an array of `HostingSlot` objects representing the completed slots (so far) in that game.

1.2.1.22 Look Up Players to Complete a Hosting Slot

The bean will provide a method to look up all the players to have completed a specific slot. The method will accept a game ID and slot ID, verify that the slot belongs to the specified game, and return an array of the IDs of those players that have completed the specified slot.

1.2.1.23 Look Up Games by Game State

The bean will provide a method to look up all games meeting specified criteria for having started and for having ended. It will accept to `Boolean` parameters, one indicating whether to include only games that have already started, only games that have not yet started, or (if null) games that of both kinds; and the other parameter indicating whether to include only games that have ended, have not ended, or (if null) games of both kinds.

1.2.1.24 Look Up Games by Player Registration

The bean will provide a method for looking up the games for which a specified player has registered. It will take a player ID as a parameter, and will return an array of the IDs of all the games (in any state) for which that player has registered.

1.2.1.25 Look Up Domains by Sponsor

The bean will provide a method for retrieving the domain information for all domains associated with a specified sponsor. It will take a sponsor ID as a parameter and will return an array of `Domain` objects containing all available details on the domains associated with that sponsor.

1.2.1.26 Look Up Hosting Slot by Domain

The bean will provide a method that finds the first slot in the hosting sequence for a particular game (established by the hosting block and hosting slot sequence numbers) that has not been completed by the specified player and that is associated with a domain whose name matches the specified string. It will return its result as a `HostingSlot`.

1.2.1.27 Retrieve All Ball Colors

The bean will provide a method that returns all the ball colors known to the component, as an array of `BallColor` objects.

1.2.2 Download Source

The component will provide a `DownloadSource` implementation (see Front Controller 2.1 documentation) that relies on the bean to retrieve downloadable data and metadata (see requirement 1.2.1.8). The simple name of this class will be `CustomDownloadSource`.

1.2.3 Database Schema

The component will interact with a database described by the following schema:

```
-- ------------------------------------------------------------
-- Contact_info entities represent the information necessary to contact
-- a user by post or telephone.
-- ------------------------------------------------------------

CREATE TABLE contact_info (
  id BIGINT NOT NULL AUTO_INCREMENT,
  first_name VARCHAR(29) NOT NULL,
  last_name VARCHAR(49) NOT NULL,
  address_1 VARCHAR(127) NOT NULL,
  address_2 VARCHAR(127) NULL,
  city VARCHAR(127) NOT NULL,
  state VARCHAR(29) NOT NULL,
  postal_code VARCHAR(9) NOT NULL,
  telephone VARCHAR(15) NOT NULL,
  PRIMARY KEY(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)
)

-- ---------------------------------------------------------
-- 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,
  suggested_name VARCHAR(255) NOT NULL,
  content BLOB NOT NULL,
  PRIMARY KEY(id)
)

-- ---------------------------------------------------------
-- This table supports the TopCoder ID Generator component,
-- version 3.0.
-- ---------------------------------------------------------

CREATE TABLE id_sequences (
  name VARCHAR(255) NOT NULL,
  next_block_start BIGINT UNSIGNED NOT NULL,
  block_size INTEGER UNSIGNED NOT NULL,
  PRIMARY KEY(name)
)

-- ---------------------------------------------------------
-- This table supports the E-mail Confirmation component used by
-- Web Registration.
-- ---------------------------------------------------------

CREATE TABLE pending_email_conf (
  address VARCHAR(255) NOT NULL,
  confirmation_code VARCHAR(32) NOT NULL,
  date_sent DATETIME NOT NULL,
  message_subject VARCHAR(255) NOT NULL,
  message_body TEXT NOT NULL,
  PRIMARY KEY(address)
)

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

CREATE TABLE any_user (
  id BIGINT NOT NULL,
  handle VARCHAR(29) NOT NULL,
  e_mail VARCHAR(255) NOT NULL,
  passwd VARCHAR(24) NOT NULL,
```

```
  is_active BOOL NOT NULL,
  PRIMARY KEY(id),
  UNIQUE INDEX ball_user_handle_unique(handle),
  UNIQUE INDEX ball_user_email_unique(e_mail)
)

-- ------------------------------------------------------------
-- 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
-- named "attributes" having string values, and named "resources"
-- having associated MIME media types, additional string data, and
-- an associated binary object.
-- ------------------------------------------------------------

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

-- ------------------------------------------------------------
-- Records counts of the number of downloads of the various
-- plugins.
-- ------------------------------------------------------------

CREATE TABLE plugin_downloads (
  plugin_name VARCHAR(255) NOT NULL AUTO_INCREMENT,
  count BIGINT NOT NULL,
  PRIMARY KEY(plugin_name)
)

-- ------------------------------------------------------------
-- Puzzle_attribute entities associate named attribute strings 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),
  INDEX puzzle_attribute_FKIndex1(puzzle_id),
  UNIQUE INDEX attribute_names_unique(puzzle_id, name),
  FOREIGN KEY(puzzle_id)
    REFERENCES puzzle(id)
)

-- ------------------------------------------------------------
-- Ball_color entities represent the available ball colors, providing a
-- color name and having an associated representative image in the form
-- of a download_obj.
-- ------------------------------------------------------------
```

```
CREATE TABLE ball_color (
  id BIGINT NOT NULL AUTO_INCREMENT,
  name VARCHAR(15) NOT NULL,
  download_obj_id BIGINT NOT NULL,
  PRIMARY KEY(id),
  INDEX ball_color_FKIndex1(download_obj_id),
  UNIQUE INDEX ball_color_name_unique(name),
  FOREIGN KEY(download_obj_id)
    REFERENCES download_obj(id)
)

-- ---------------------------------------------------------
-- Admin entities represent application administrators.  They have
-- associated user credentials, but nothing else (in this version).
-- ---------------------------------------------------------

CREATE TABLE admin (
  any_user_id BIGINT NOT NULL,
  PRIMARY KEY(any_user_id),
  INDEX admin_FKIndex1(any_user_id),
  FOREIGN KEY(any_user_id)
    REFERENCES any_user(id)
)

-- ---------------------------------------------------------
-- Game entities represent individual Barooka Ball games.  They have
-- a serial number (in the form of their ID), a ball color, and
nullable start
-- and end dates.
-- ---------------------------------------------------------

CREATE TABLE game (
  id BIGINT NOT NULL AUTO_INCREMENT,
  ball_color_id BIGINT NOT NULL,
  start_date DATETIME NOT NULL,
  keys_required INTEGER UNSIGNED NOT NULL,
  PRIMARY KEY(id),
  INDEX game_FKIndex1(ball_color_id),
  FOREIGN KEY(ball_color_id)
    REFERENCES ball_color(id)
)

-- ---------------------------------------------------------
-- Sponsor entities represent game sponsor representatives
-- eligible to place bids to host the application.  They have user
-- credentials, standard contact info, plus an optional (nullable)
-- FAX number, and a payment method preference (for making payments).
-- ---------------------------------------------------------

CREATE TABLE sponsor (
  any_user_id BIGINT NOT NULL,
  contact_info_id BIGINT NOT NULL,
  fax NUMERIC NULL,
```

```
  payment_pref VARCHAR(49) NULL,
  is_approved BOOL NULL,
  PRIMARY KEY(any_user_id),
  INDEX sponsor_FKIndex2(contact_info_id),
  INDEX sponsor_FKIndex2(any_user_id),
  FOREIGN KEY(contact_info_id)
    REFERENCES contact_info(id)
  FOREIGN KEY(any_user_id)
    REFERENCES any_user(id)
)

-- ---------------------------------------------------------
-- Puzzle_resource entities associate arbitrary binary objects with
-- specific puzzle entities.  They associate a name with MIME media
-- type information, and a corresponding binary object
-- ---------------------------------------------------------

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),
  INDEX puzzle_resource_FKIndex1(puzzle_id),
  INDEX puzzle_resource_FKIndex2(download_obj_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)
)

-- ---------------------------------------------------------
-- Player entities represent players.  They have user credentials,
-- and may have associated contact information and payment preference
-- (for being paid prize money).
-- ---------------------------------------------------------

CREATE TABLE player (
  any_user_id BIGINT NOT NULL,
  contact_info_id BIGINT NULL,
  payment_pref VARCHAR(49) NULL,
  PRIMARY KEY(any_user_id),
  INDEX player_FKIndex2(contact_info_id),
  INDEX player_FKIndex3(any_user_id),
  FOREIGN KEY(contact_info_id)
    REFERENCES contact_info(id)
  FOREIGN KEY(any_user_id)
    REFERENCES any_user(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),
  INDEX game_registration_FKIndex1(player_id),
  INDEX game_registration_FKIndex2(game_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),
  INDEX provisional_winner_FKIndex1(game_id),
  INDEX provisional_winner_FKIndex2(player_id),
  FOREIGN KEY(game_id)
    REFERENCES game(id)
  FOREIGN KEY(player_id)
    REFERENCES player(any_user_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,
  sequence_number INTEGER UNSIGNED NOT NULL,
  max_time_per_slot INTEGER UNSIGNED NOT NULL,
  PRIMARY KEY(id),
  INDEX hosting_block_FKIndex2(game_id),
  UNIQUE INDEX unique_game_block_seq(game_id, sequence_number),
  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),
  INDEX auction_FKIndex1(hosting_block_id),
  FOREIGN KEY(hosting_block_id)
    REFERENCES hosting_block(id)
)

-- ---------------------------------------------------------------
-- Domain entities represent web sites that (it is hoped) are
-- eligible to host the application.  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),
  INDEX domain_FKIndex1(sponsor_id),
  UNIQUE INDEX dmn_spn_url_unique(sponsor_id, 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),
  INDEX plyr_won_game_FKIndex1(player_id),
  INDEX plyr_won_game_FKIndex2(game_id),
  FOREIGN KEY(player_id)
    REFERENCES player(any_user_id)
  FOREIGN KEY(game_id)
```

```
    REFERENCES game(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),
  INDEX image_FKIndex1(domain_id),
  INDEX image_FKIndex2(download_obj_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 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),
  INDEX bid_FKIndex2(image_id),
  INDEX bid_FKIndex2(auction_id),
  FOREIGN KEY(auction_id)
    REFERENCES auction(hosting_block_id)
  FOREIGN KEY(image_id)
    REFERENCES image(id)
)
```

```
-- ------------------------------------------------------------
-- Hosting_slot entities represent specific appearances of the Ball 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),
  UNIQUE INDEX hosting_slot_unique_bid(bid_id),
  FOREIGN KEY(bid_id)
    REFERENCES bid(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),
  INDEX Table_27_FKIndex1(bid_id),
  FOREIGN KEY(bid_id)
    REFERENCES bid(id)
)

-- ------------------------------------------------------------
-- The puzzle_for_slot relation associates the appropriate game-win
-- puzzle with a particular hosting slot.
-- ------------------------------------------------------------

CREATE TABLE puzzle_for_slot (
  hosting_slot_id BIGINT NOT NULL,
  puzzle_id BIGINT NOT NULL,
  PRIMARY KEY(hosting_slot_id),
  INDEX game_win_puzzle_FKIndex1(hosting_slot_id),
  INDEX game_win_puzzle_FKIndex2(puzzle_id),
  FOREIGN KEY(hosting_slot_id)
    REFERENCES hosting_slot(id)
```

```
  FOREIGN KEY(puzzle_id)
    REFERENCES puzzle(id)
)

-- ------------------------------------------------------------
-- Target_object entities represent the identifiers of the target
-- objects in the chain of links associated with a specific hosting
-- slot.  Each one has a sequence number within the chain and a
-- hex-encoded hash of the target identifier, plus a path (relative
-- to the host domain) to the location of the target and the ID of
-- the clue image to provide for this target
-- ------------------------------------------------------------

CREATE TABLE target_object (
  id BIGINT NOT NULL AUTO_INCREMENT,
  hosting_slot_id BIGINT NOT NULL,
  sequence_number INTEGER UNSIGNED NOT NULL,
  uri_path VARCHAR(255) NOT NULL,
  identifier_text VARCHAR(20) NULL,
  identifier_hash VARCHAR(40) NOT NULL,
  clue_img_id BIGINT NOT NULL,
  PRIMARY KEY(id),
  INDEX target_object_FKIndex1(hosting_slot_id),
  UNIQUE INDEX slot_seq_unique(hosting_slot_id, sequence_number),
  INDEX target_object_FKIndex2(clue_img_id),
  FOREIGN KEY(hosting_slot_id)
    REFERENCES hosting_slot(id)
  FOREIGN KEY(clue_img_id)
    REFERENCES download_obj(id)
)

-- ------------------------------------------------------------
-- The brn_tsr_for_slot relation associates hosting slots with their
-- brain teasers.
-- ------------------------------------------------------------

CREATE TABLE brn_tsr_for_slot (
  hosting_slot_id BIGINT NOT NULL,
  sequence_number INTEGER UNSIGNED NOT NULL,
  puzzle_id BIGINT NOT NULL,
  PRIMARY KEY(hosting_slot_id, sequence_number),
  INDEX brain_teaser_FKIndex1(hosting_slot_id),
  INDEX brain_teaser_FKIndex2(puzzle_id),
  FOREIGN KEY(hosting_slot_id)
    REFERENCES hosting_slot(id)
  FOREIGN KEY(puzzle_id)
    REFERENCES puzzle(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),
  INDEX slot_completion_FKIndex1(player_id),
  INDEX slot_completion_FKIndex2(hosting_slot_id),
  INDEX plyr_compltd_slot_FKIndex3(key_image_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)
)
```

1.2.4 Internal Interfaces

1.2.4.1 Data Transfer Objects

The component will provide the following data transfer object interfaces, and will provide implementations for its own use:

```
package com.orpheus.game.persistence;

/**
 * The Game interface represents an individual game managed by the
 * application. It carries a unique identifier, a start date, and an
 * end date, and can provide a BallColor representing the color
 * associated with this game and a game name string computed based on
 * the game id and color.
*/
public interface Game extends java.io.Serializable {

      /**
       * Gets the identifier for this game, as a Long. The identifier
       * may be null if this Game has not yet been assigned one.
       *
       * @return
       */
      public Long getId();

      /**
       * Gets the name of this game, which is the concatenation of the
       * name of the associated BallColor with the ID of this game. If
       * this game does not have an ID or BallColor yet assigned then
       * that part of the name is represented by a single question
       * mark.
       *
       * @return
       */
      public String getName();
```

```
        /**
         * Returns the BallColor object assigned to this game, or null if
         * there is none.
         *
         * @return
         */
        public BallColor getBallColor();

        /**
         * Returns the number of keys required to attempt to win this
         * game
         *
         * @return
         */
        public int getKeyCount();

        /**
         * Retrieves the planned or past start date for this game; will
         * not be null.
         *
         * @return
         */
        public java.util.Date getStartDate();

        /**
         * Retrieves the end date of this game, if it has already ended,
         * or null if it hasn't.
         *
         * @return
         */
        public java.util.Date getEndDate();

        /**
         * Retrieves an array of HostingBlock objects representing the
         * hosting blocks within this game
         *
         * @return
         */
        public HostingBlock[] getBlocks();
}

/**
 * A BallColor object represents a supported Barooka Ball color
 */
public interface BallColor extends java.io.Serializable {

        /**
         * Returns the unique ID of this BallColor
         *
         * @return
         */
        public Long getId();
```

```
        /**
         * Gets the color name, such as "RED", "BLUE", or "TANGERINE".
         *
         * @return
         */
        public String getName();

        /**
         * Returns the ID of a downloadable object that contains the ball
         * image corresponding to this BallColor.
         *
         * @return
         */
        public long getImageId();
    }

    /**
     * Represents a 'block'; of hosting slots. Blocks serve as an
     * organizational unit for hosting auctions, and furthermore help to
     * obscure the specific sequence of upcoming domains, even from
     * sponsors privy to the auction details.
     */
    public interface HostingBlock extends java.io.Serializable {

        /**
         * Returns the ID of this block as a Long, or null if no ID has
         * yet been assigned.
         *
         * @return
         */
        public Long getId();

        /**
         * Returns the sequence number of this block within its game
         *
         * @return
         */
        public int getSequenceNumber();

        /**
         * Returns an array of HostingSlot objects representing all the
         * slots associated with this block
         *
         * @return
         */
        public HostingSlot[] getSlots();

        /**
         * Returns the maximum hosting time for this block, in minutes
         *
         * @return
         */
        public int getMaxHostingTimePerSlot();
    }
```

```
/**
 * An interface representing the persistent information about a
 * particular hosting slot
 */
public interface HostingSlot extends java.io.Serializable {

      /**
       * Gets the ID of this hosting slot, or null if none has yet been
       * assigned.
       *
       * @return
       */
      public Long getId();

      /**
       * Returns a Domain object represented the domain assigned to
       * this hosting slot
       *
       * @return
       */
      public Domain getDomain();

      /**
       * Returns the ID of the image information associated with this
       * hosting slot
       *
       * @return
       */
      public long getImageId();

      /**
       * Returns the unique IDs of the brain teasers in this slot's
       * brain teaser series
       *
       * @return
       */
      public long[] getBrainTeaserIds();

      /**
       * Returns the ID of the puzzle assigned to this slot, or null if
       * there is none
       *
       * @return
       */
      public Long getPuzzleId();

      /**
       * Returns the sequence number of this slot within its block
       *
       * @return
       */
      public int getSequenceNumber();
```

```
        /**
         * Returns an array of DomainTarget objects representing the
         * 'minihunt targets'; for this hosting slot
         *
         * @return
         */
        public DomainTarget[] getDomainTargets();

        /**
         * Returns the amount of the winning bid in the auction for this
         * slot
         *
         * @return
         */
        public int getWinningBid();

        /**
         * Returns a Date representing the date and time at which this
         * hosting slot began hosting, or null if it has not yet started
         * hosting
         *
         * @return
         */
        public java.util.Date getHostingStart();

        /**
         * Returns a Date representing the date and time at which this
         * hosting slot stopped hosting, or null if it has not yet
         * stopped (including if it hasn't begun)
         *
         * @return
         */
        public java.util.Date getHostingEnd();
}

/**
 * An interface representing a hosting domain within the application
 */
public interface Domain extends java.io.Serializable {

        /**
         * Returns the unique ID for this domain, or null if none has yet
         * been assigned
         *
         * @return
         */
        public Long getId();

        /**
         * Returns the user ID number of the sponsor to whom this domain
         * is assigned
         *
         * @return
```

```
         */
        public long getSponsorId();

        /**
         * Returns the name of this domain -- i.e. the DNS name of the
         * host -- as a String
         *
         * @return
         */
        public String getDomainName();

        /**
         * Returns the value of this domain's approval flag, or null if
         * no approval decision has been made
         *
         * @return
         */
        public Boolean isApproved();

        /**
         * Returns ImageInfo objects representing all the images
         * associated with this domain
         *
         * @return
         */
        public ImageInfo[] getImages();

}

/**
 * An interface representing the stored information about an image
 * associated with a specific domain
 */
public interface ImageInfo extends java.io.Serializable {

        /**
         * Returns the unique ID associated with this image information,
         * or null if none has yet been assigned
         *
         * @return
         */
        public Long getId();

        /**
         * Returns the unique ID of the downloadable image data
         * associated with this image information
         *
         * @return
         */
        public long getDownloadId();

        /**
         * Returns a String description of the image
         *
```

```
        * @return
        */
       public String getDescription();

       /**
        * Returns the value of the approval flag for this image, or null
        * if no approval decision has yet been made
        *
        * @return
        */
       public Boolean isApproved();
}

/**
 * Represents an object to be sought by players on a host site.
 */
public interface DomainTarget extends java.io.Serializable {

       /**
        * The unique identifier of this target, or null if none has yet
        * been assigned
        *
        * @return
        */
       public Long getId();

       /**
        * Returns the sequence number of this target among those
        * assigned to the same hosting slot
        *
        * @return
        */
       public int getSequenceNumber();

       /**
        * Returns the path and file parts of the URI at which the target
        * is located
        *
        * @return
        */
       public String getUriPath();

       /**
        * Returns the plain text identifier of the target
        *
        * @return
        */
       public String getIdentifierText();

       /**
        * Returns a hash of the target's identifier
        *
        * @return
        */
```

```
        public String getIdentifierHash();

        /**
         * Returns the unique identifier of a downloadable object
         * constituting an image to be presented to users as the clue for
         * this target
         *
         * @return
         */
        public long getClueImageId();
}


/**
 * Represents the recorded data about a player's completion of a
 * hosting slot.
 */
public interface SlotCompletion {

        /**
         * Returns the ID of the hosting slot that was completed
         *
         * @return
         */
        public long getSlotId();

        /**
         * Returns the ID of the player who completed the slot
         *
         * @return
         */
        public long getPlayerId();

        /**
         * Returns a Date representing the date and time at which the
         * slot was completed
         *
         * @return
         */
        public java.util.Date getTimestamp();

        /**
         * Returns the text of the key associated with this completion
         *
         * @return
         */
        public String getKeyText();

        /**
         * Returns the download object ID of an image containing the key
         * text, to be presented to users instead of plain text.
         *
         * @return
         */
```

```
        public long getKeyImageId();
    }
```

1.2.4.2 EJB Interfaces

The component will provide the following home, local home, component, and local component interfaces for the provided game data EJB (along with a suitable bean implementation class; see requirement 1.2.1):

```
    package com.orpheus.game.persistence;

    /**
     * The local home interface of the GameData EJB
     */
    public interface GameDataHome extends javax.ejb.EJBHome {

        /**
         * Obtains an instance of the GameData bean
         *
         * @return
         * @throws CreateException if the bean container is unable to
         *    allocate a bean instance to service the request
         * @throws RemoteException if a communication error occurs
         *    between client and EJB container
         */
        public GameData create() throws javax.ejb.CreateException,
            java.rmi.RemoteException;
    }

    /**
     * The (remote) home interface of the GameData EJB
     */
    public interface GameDataLocalHome extends javax.ejb.EJBLocalHome {

        /**
         * Obtains an instance of the GameDataLocal bean
         *
         * @return
         * @throws CreateException if the bean container is unable to
         *    allocate a bean instance to service the request
         */
        public GameDataLocal create() throws javax.ejb.CreateException;
    }

    /**
     * The (remote) component interface of the GameData EJB, which provides
     * access to persistent information about games managed by the
     * application
     */
    public interface GameData extends javax.ejb.EJBObject {

        /**
         * Creates a new game entity in the persistent store, along with
         * associated hosting blocks. Any game or block IDs that are null
```

```
     * will be automatically assigned acceptable values. No hosting
     * slots are created for the game at this time. The returned Game
     * object will represent the persisted data, including any IDs
     * assigned to the game and blocks.
     *
     * @param game
     * @return
     */
    public Game createGame(Game game)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Creates hosting slots associates with the specified Bid IDs in
     * the specified hosting block
     *
     * @param blockId
     * @param bidIds
     * @return
     */
    public HostingSlot[] createSlots(long blockId, long[] bidIds)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Creates a new persistent domain representation with the data
     * from the provided Domain object and its nested ImageInfo
     * objects. Any null Domain or ImageIndo IDs are assigned
     * appropriate values. The returned Domain will reflect the
     * persistent representation, including any automatically
     * assigned IDs.
     *
     * @param domain
     * @return
     */
    public Domain createDomain(Domain domain)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Creates a new, persistent hosting block for the specified
     * game. The block will having an auto-assigned ID, the next
     * available sequence number after those of the game's existing
     * blocks (or 1 if there are no other blocks), no hosting slots,
     * and the specified maximum hosting time per slot. It returns a
     * HostingBlock object representing the new block.
     *
     * @param gameId
     * @param slotMaxHostingTime
     * @return
     */
    public HostingBlock addBlock(long gameId, int slotMaxHostingTime)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;
```

```
/**
 * Retrieves a Game object representing the Game having the
 * specified ID
 *
 * @param gameId
 * @return
 */
public Game getGame(long gameId) throws java.rmi.RemoteException,
      com.orpheus.game.GameDataException;

/**
 * Retrieves a HostingBlock object representing the hosting block
 * having the specified ID
 *
 * @param blockId
 * @return
 */
public HostingBlock getBlock(long blockId)
      throws java.rmi.RemoteException,
      com.orpheus.game.GameDataException;

/**
 * Retrieves a HostingSlot object reqpresenting the slot having
 * the specified ID
 *
 * @param slotId
 * @return
 */
public HostingSlot getSlot(long slotId)
      throws java.rmi.RemoteException,
      com.orpheus.game.GameDataException;

/**
 * Retrieves the DownloadData object corresponding to the
 * specified ID
 *
 * @param id
 * @return
 */
public com.topcoder.web.frontcontroller.results.DownloadData
      getDownloadData(long id) throws java.rmi.RemoteException,
      com.orpheus.game.GameDataException;

/**
 * Retrieves a Domain object representing the domain
 * corresponding to the specified ID
 *
 * @param domainId
 * @return
 */
public Domain getDomain(long domainId)
      throws java.rmi.RemoteException,
      com.orpheus.game.GameDataException;
```

```
/**
 * Returns the key text for the specified player's completions of
 * the specified slots. The length of the returned array is the
 * same as the length of the slotIds argument, and their elements
 * correspond: each string in the returned array is the key text
 * associated with the slot completion by the specified player of
 * the slot whose ID appears at the same index in the input
 * slotIds. If the specified player has not completed any
 * particular slot specified among the slot IDs then the
 * corresponding element or the returned array is null.
 *
 * @param playerId
 * @param slotIds
 * @return
 */
public String[] getKeysForPlayer(long playerId, long[] slotIds)
      throws java.rmi.RemoteException,
      com.orpheus.game.GameDataException;

/**
 * Retrieves the PuzzleData associated with the specified puzzle
 * ID.
 *
 * @param puzzleId
 * @return
 */
public com.topcoder.util.puzzle.PuzzleData getPuzzle(
      long puzzleId) throws java.rmi.RemoteException,
      com.orpheus.game.GameDataException;

/**
 * Increments the download count for the plugin identified by the
 * specified name
 *
 * @param pluginName
 */
public void recordPluginDownload(String pluginName)
      throws java.rmi.RemoteException,
      com.orpheus.game.GameDataException;

/**
 * Records the specified player's registration for the specified
 * game
 *
 * @param playerId
 * @param gameId
 */
public void recordRegistration(long playerId, long gameId)
      throws java.rmi.RemoteException,
      com.orpheus.game.GameDataException;

/**
 * Records the completion of the specified slot by the specified
```

```
     * player at the specified date and time, and generates a key for
     * the playe to associate with the completion.
     *
     * @param playerId
     * @param slotId
     * @param date
     * @return
     */
    public SlotCompletion recordSlotCompletion(long playerId,
          long slotId, java.util.Date date)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Records the fact that the specified player has completed the
     * specified game. Whether or not such a player actually wins the
     * game depends on whether others have already completed the
     * game, and on administrative establishment of winner
     * eligibility.
     *
     * @param playerId
     * @param gameId
     */
    public void recordGameCompletion(long playerId, long gameId)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Records a binary object in the database, such as might later
     * be retrieved by the custom DownloadSource. The ID assigned to
     * the binary object is returned.
     *
     * @param name
     * @param mediaType
     * @param content
     * @return
     */
    public long recordBinaryObject(String name, String mediaType,
          byte[] content) throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Updates the persistent hosting slot information for the
     * existing slots represented by the specified HostingSlot
     * objects, so that the persistent representation matches the
     * provided objects. Nested DomainTarget objects may or may not
     * already be recorded in persistence; the component assumes that
     * DomainTarget's with null IDs are new, and that others already
     * exist in the database. The component will assign IDs for new
     * DomainTargets as needed.
     *
     * This method will also update the following additional
     * HostingSlot properties (only): sequence number, hosting start,
     * hosting end, brain teaser IDs, puzzle ID.  It will return an
```

```
     * array containing the revised hosting slots.
     *
     * @param slots
     * @return
     */
    public HostingSlot[] updateSlots(HostingSlot[] slots)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Updates the persistent domain information for the specified
     * Domain object to match the Domain object, where the
     * appropriate persistent record is identified by the Domain's ID
     *
     * @param domain
     */
    public void updateDomain(Domain domain)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Deletes the hosting slot having the specified ID
     *
     * @param slotId
     */
    public void deleteSlot(long slotId)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Looks up all distinct domains hosting any slot in any active
     * game, and returns an array of Domain objects representing the
     * results
     *
     * @return
     */
    public Domain[] findActiveDomains()
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Looks up all ongoing games in which a domain matching the
     * specified string is a host in a slot that the specified player
     * has not yet completed, and returns an array of all such games
     *
     * @param playerId
     * @return
     */
    public Game[] findGamesByDomain(String domain, long playerId)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Looks up all hosting slots completed by any player in the
```

```
     * specified game, and returns the results as an array of
     * HostingSlot objects. Returned slots are in ascending order by
     * first completion time, or equivalently, in ascending order by
     * hosting block sequence number and hosting slot sequence
     * number.
     *
     * @param gameId
     * @return
     */
    public HostingSlot[] findCompletedSlots(long gameId)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Looks up all the players who are recorded as having completed
     * the specified hosting slot in the specified game, and returns
     * an array of their IDs.
     *
     * @param gameId
     * @param slotId
     * @return
     */
    public long[] findSlotCompletingPlayers(long gameId, long slotId)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Retrieves game information for games meeting the specified
     * game status criteria
     *
     * @param isStarted a Boolean having value true to restrict to
     *    games that have started or false to restrict to games that
     *    have not yet started; null to ignore whether games have
     *    started
     * @param isEnded a Boolean having value true to restrict to
     *    games that have ended or false to restrict to games that
     *    have not yet ended; null to ignore whether games have ended
     * @return an array of Game objects representing the games found
     */
    public Game[] findGames(Boolean isStarted, Boolean isEnded)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;

    /**
     * Looks up all the games for which the specified player is
     * registered, and returns an array of their IDs
     *
     * @param playerId
     * @return
     */
    public long[] findGameRegistrations(long playerId)
          throws java.rmi.RemoteException,
          com.orpheus.game.GameDataException;
```

```
        /**
         * Looks up all domains associated with the specified sponsor and
         * returns an array of Domain objects representing them
         *
         * @param sponsorId
         * @return
         */
        public Domain[] findDomainsForSponsor(long sponsorId)
              throws java.rmi.RemoteException,
              com.orpheus.game.GameDataException;

        /**
         * Finds the first HostingSlot in the hosting sequence for the
         * specified game that is assigned the specified domain and has
         * not yet been completed by the specified player.
         *
         * @param gameId
         * @param playerId
         * @param domain
         * @return
         */
        public HostingSlot findSlotForDomain(long gameId, long playerId,
              String domain) throws java.rmi.RemoteException,
              com.orpheus.game.GameDataException;

        /**
         * Provides information about all ball colors available to the
         * application.
         *
         * @return
         */
        public BallColor[] findAllBallColors()
              throws java.rmi.RemoteException,
              com.orpheus.game.GameDataException;
}

/**
 * The local component interface of the GameData EJB, which provides
 * access to persistent information about games managed by the
 * application
 */
public interface GameDataLocal extends javax.ejb.EJBLocalObject {

        /**
         * Creates a new game entity in the persistent store, along with
         * associated hosting blocks. Any game or block IDs that are null
         * will be automatically assigned acceptable values. No hosting
         * slots are created for the game at this time. The returned Game
         * object will represent the persisted data, including any IDs
         * assigned to the game and blocks.
         *
         * @param game
         * @return
         */
```

```
public Game createGame(Game game)
      throws com.orpheus.game.GameDataException;

/**
 * Creates hosting slots associates with the specified Bid IDs in
 * the specified hosting block
 *
 * @param blockId
 * @param bidIds
 * @return
 */
public HostingSlot[] createSlots(long blockId, long[] bidIds)
      throws com.orpheus.game.GameDataException;

/**
 * Creates a new persistent domain representation with the data
 * from the provided Domain object and its nested ImageInfo
 * objects. Any null Domain or ImageIndo IDs are assigned
 * appropriate values. The returned Domain will reflect the
 * persistent representation, including any automatically
 * assigned IDs.
 *
 * @param domain
 * @return
 */
public Domain createDomain(Domain domain)
      throws com.orpheus.game.GameDataException;

/**
 * Creates a new, persistent hosting block for the specified
 * game. The block will having an auto-assigned ID, the next
 * available sequence number after those of the game's existing
 * blocks (or 1 if there are no other blocks), no hosting slots,
 * and the specified maximum hosting time per slot. It returns a
 * HostingBlock object representing the new block.
 *
 * @param gameId
 * @param slotMaxHostingTime
 * @return
 */
public HostingBlock addBlock(long gameId, int slotMaxHostingTime)
      throws com.orpheus.game.GameDataException;

/**
 * Retrieves a Game object representing the Game having the
 * specified ID
 *
 * @param gameId
 * @return
 */
public Game getGame(long gameId)
      throws com.orpheus.game.GameDataException;

/**
```

```
     * Retrieves a HostingBlock object representing the hosting block
     * having the specified ID
     *
     * @param blockId
     * @return
     */
    public HostingBlock getBlock(long blockId)
          throws com.orpheus.game.GameDataException;

    /**
     * Retrieves a HostingSlot object reqpresenting the slot having
     * the specified ID
     *
     * @param slotId
     * @return
     */
    public HostingSlot getSlot(long slotId)
          throws com.orpheus.game.GameDataException;

    /**
     * Retrieves the DownloadData object corresponding to the
     * specified ID
     *
     * @param id
     * @return
     */
    public com.topcoder.web.frontcontroller.results.DownloadData
          getDownloadData(long id)
          throws com.orpheus.game.GameDataException;

    /**
     * Retrieves a Domain object representing the domain
     * corresponding to the specified ID
     *
     * @param domainId
     * @return
     */
    public Domain getDomain(long domainId)
          throws com.orpheus.game.GameDataException;

    /**
     * Returns the key text for the specified player's completions of
     * the specified slots. The length of the returned array is the
     * same as the length of the slotIds argument, and their elements
     * correspond: each string in the returned array is the key text
     * associated with the slot completion by the specified player of
     * the slot whose ID appears at the same index in the input
     * slotIds. If the specified player has not completed any
     * particular slot specified among the slot IDs then the
     * corresponding element or the returned array is null.
     *
     * @param playerId
     * @param slotIds
     * @return
```

```
     */
    public String[] getKeysForPlayer(long playerId, long[] slotIds)
          throws com.orpheus.game.GameDataException;

    /**
     * Retrieves the PuzzleData associated with the specified puzzle
     * ID.
     *
     * @param puzzleId
     * @return
     */
    public com.topcoder.util.puzzle.PuzzleData getPuzzle(
          long puzzleId) throws com.orpheus.game.GameDataException;

    /**
     * Increments the download count for the plugin identified by the
     * specified name
     *
     * @param pluginName
     */
    public void recordPluginDownload(String pluginName)
          throws com.orpheus.game.GameDataException;

    /**
     * Records the specified player's registration for the specified
     * game
     *
     * @param playerId
     * @param gameId
     */
    public void recordRegistration(long playerId, long gameId)
          throws com.orpheus.game.GameDataException;

    /**
     * Records the completion of the specified slot by the specified
     * player at the specified date and time, and generates a key for
     * the playe to associate with the completion.
     *
     * @param playerId
     * @param slotId
     * @param date
     * @return
     */
    public SlotCompletion recordSlotCompletion(long playerId,
          long slotId, java.util.Date date)
          throws com.orpheus.game.GameDataException;

    /**
     * Records the fact that the specified player has completed the
     * specified game. Whether or not such a player actually wins the
     * game depends on whether others have already completed the
     * game, and on administrative establishment of winner
     * eligibility.
     *
```

```
     * @param playerId
     * @param gameId
     */
    public void recordGameCompletion(long playerId, long gameId)
          throws com.orpheus.game.GameDataException;

    /**
     * Records a binary object in the database, such as might later
     * be retrieved by the custom DownloadSource. The ID assigned to
     * the binary object is returned.
     *
     * @param name
     * @param mediaType
     * @param content
     * @return
     */
    public long recordBinaryObject(String name, String mediaType,
          byte[] content) throws com.orpheus.game.GameDataException;

    /**
     * Updates the persistent hosting slot information for the
     * existing slots represented by the specified HostingSlot
     * objects, so that the persistent representation matches the
     * provided objects. Nested DomainTarget objects may or may not
     * already be recorded in persistence; the component assumes that
     * DomainTarget's with null IDs are new, and that others already
     * exist in the database. The component will assign IDs for new
     * DomainTargets as needed.
     *
     * This method will also update the following additional
     * HostingSlot properties (only): sequence number, hosting start,
     * hosting end, brain teaser IDs, puzzle ID.  It will return an
     * array containing the revised hosting slots.
     *
     * @param slots
     * @return
     */
    public HostingSlot[] updateSlots(HostingSlot[] slots)
          throws com.orpheus.game.GameDataException;

    /**
     * Updates the persistent domain information for the specified
     * Domain object to match the Domain object, where the
     * appropriate persistent record is identified by the Domain's ID
     *
     * @param domain
     */
    public void updateDomain(Domain domain)
          throws com.orpheus.game.GameDataException;

    /**
     * Deletes the hosting slot having the specified ID
     *
     * @param slotId
```

```
     */
    public void deleteSlot(long slotId)
          throws com.orpheus.game.GameDataException;

    /**
     * Looks up all distinct domains hosting any slot in any active
     * game, and returns an array of Domain objects representing the
     * results
     *
     * @return
     */
    public Domain[] findActiveDomains()
          throws com.orpheus.game.GameDataException;

    /**
     * Looks up all ongoing games in which a domain matching the
     * specified string is a host in a slot that the specified player
     * has not yet completed, and returns an array of all such games
     *
     * @param playerId
     * @return
     */
    public Game[] findGamesByDomain(String domain, long playerId)
          throws com.orpheus.game.GameDataException;

    /**
     * Looks up all hosting slots completed by any player in the
     * specified game, and returns the results as an array of
     * HostingSlot objects. Returned slots are in ascending order by
     * first completion time, or equivalently, in ascending order by
     * hosting block sequence number and hosting slot sequence
     * number.
     *
     * @param gameId
     * @return
     */
    public HostingSlot[] findCompletedSlots(long gameId)
          throws com.orpheus.game.GameDataException;

    /**
     * Looks up all the players who are recorded as having completed
     * the specified hosting slot in the specified game, and returns
     * an array of their IDs.
     *
     * @param gameId
     * @param slotId
     * @return
     */
    public long[] findSlotCompletingPlayers(long gameId, long slotId)
          throws com.orpheus.game.GameDataException;

    /**
     * Retrieves game information for games meeting the specified
     * game status criteria
```

```
     *
     * @param isStarted a Boolean having value true to restrict to
     *    games that have started or false to restrict to games that
     *    have not yet started; null to ignore whether games have
     *    started
     * @param isEnded a Boolean having value true to restrict to
     *    games that have ended or false to restrict to games that
     *    have not yet ended; null to ignore whether games have ended
     * @return an array of Game objects representing the games found
     */
    public Game[] findGames(Boolean isStarted, Boolean isEnded)
          throws com.orpheus.game.GameDataException;

    /**
     * Looks up all the games for which the specified player is
     * registered, and returns an array of their IDs
     *
     * @param playerId
     * @return
     */
    public long[] findGameRegistrations(long playerId)
          throws com.orpheus.game.GameDataException;

    /**
     * Looks up all domains associated with the specified sponsor and
     * returns an array of Domain objects representing them
     *
     * @param sponsorId
     * @return
     */
    public Domain[] findDomainsForSponsor(long sponsorId)
          throws com.orpheus.game.GameDataException;

    /**
     * Finds the first HostingSlot in the hosting sequence for the
     * specified game that is assigned the specified domain and has
     * not yet been completed by the specified player.
     *
     * @param gameId
     * @param playerId
     * @param domain
     * @return
     */
    public HostingSlot findSlotForDomain(long gameId, long playerId,
          String domain) throws com.orpheus.game.GameDataException;

    /**
     * Provides information about all ball colors available to the
     * application.
     *
     * @return
     */
    public BallColor[] findAllBallColors()
          throws com.orpheus.game.GameDataException;
```

```
}
```

1.2.5 Object Property to Database Field Relationships

Object Property	Database Realization	Comments
Game.id	game.id	
Game.name	n/a	a concatenation of BallColor.name and Game.id
Game.ballColor	(a row from ball_color)	games are associated with ball_colors through game.ball_color_id
Game.keyCount	game.keys_required	
Game.startDate	game.start_date	
Game.endDate	plyr_won_game.date	for the plyr_won_game row with the corresponding game_id, if any
Game.blocks	(rows from hosting_block)	those rows with the corresponding game_id
BallColor.id	ball_color.id	
BallColor.name	ball_color.name	
BallColor.imageId	ball_color.download_obj_id	
HostingBlock.id	hosting_block.id	
HostingBlock.sequenceNumber	hosting_block.sequence_number	
HostingBlock.slots	(rows from hosting_slot)	hosting_slots are associated with bids, which are associated with auctions, which are associated with hosting_blocks
HostingBlock.maxHostingTimePerSlot	hosting_block.max_time_per_slot	
HostingSlot.id	hosting_slot.id	
HostingSlot.sequenceNumber	hosting_slot.sequence_number	
HostingSlot.start	hosting_slot.hosting_start	
HostingSlot.end	hosting_slot.hosting_end	
HostingSlot.domain	(a row from domain)	hosting_slots are associated with bids, which are associated with images, which are directly associated with domains
HostingSlot.imageId	bid.image_id	for the bid identified by hosting_slot.bid_id
HostingSlot.brainTeaserIds	multiple values from brn_tsr_for_slot.puzzle_id	where brn_tsr_for_slot.hosting_slot_id corresponds to the slot, and brn_tsr_for_slot.sequence_number determines the order of the IDs

HostingSlot.puzzleId	puzzle_for_slot.puzzle_id	where puzzle_for_slot.hosting_slot_id corresponds to the slot
HostingSlot.domainTargets	(rows from target_object)	where target_object.hosting_slot_id corresponds to the slot, and target_object.sequence_number determines the order
HostingSlot.winningBid	effective_bid.current_amount	hosting_slots are associated with bids, which are directly associated with effective_bids
Domain.id	domain.id	
Domain.sponsorId	domain.sponsor_id	
Domain.domainName	domain.base_url	
Domain.isApproved	domain.is_approved	
Domain.images	(rows from image)	where image.domain_id associates images with their domains
DomainTarget.id	target_object.id	
DomainTarget.sequenceNumber	target_object.sequence_number	
DomainTarget.uriPath	target_object.uri_path	
DomainTarget.identifierText	target_object.identifier_text	
DomainTarget.identifierHash	target_object.identifier_hash	
DomainTarget.clueImageId	target_object.clue_img_id	
ImageInfo.id	image.id	
ImageInfo.downloadId	image.download_obj_id	
ImageInfo.description	image.description	
ImageInfo.isApproved	image.is_approved	
SlotCompletion.slotId	plyr_compltd_slot.hosting_slot_id	
SlotCompletion.playerId	plyr_compltd_slot.player_id	
SlotCompletion.timestamp	plyr_compltd_slot.timestamp	
SlotCompletion.keyText	plyr_compltd_slot.key_text	
SlotCompletion.keyImageId	plyr_compltd_slot.key_image_id	

1.3 Required Algorithms

None

1.4 Example of the Software Usage

The component will be used to provide access to game 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 DownloadSource / DownloadData

The component provides an implementation of this interface from the Front Controller 2.1 component:

```
package com.topcoder.frontcontroller.results;

/**
 * Defines the behavior of pluggable sources of downloadable data and
 * associated metadata.
 */
public interface DownloadSource {

      /**
       * Determines and returns the download data and metadata
       * associated with the specified ID string, or returns null if it
       * cannot associate a download with the specified ID.
       *
       * @throws HandlerExecutionException if this method fails because
       * of an internal checked exception
       */
      public DownloadData getDownloadData(String id)
                  throws HandlerExecutionException;
}
```

The component may use the `DownloadData` implementation provided with the Front Controller; for reference, however, here is the definition of the `DownloadData` interface:

```
/**
 * Represents a complete, downloadable entity with associated media
 * type and (possibly) suggested file name
 */
public interface DownloadData {

      /**
       * Returns an InputStream from which this entity's content can be
       * read.  The stream is initially positioned at the first byte of
       * the download data, and the download data continue to end of
       * stream.
       */
      public java.io.InputStream getContent();

      /**
       * Returns the media type of the download data, in MIME format.
```

```
         */
        public String getMediaType();

        /**
         * Returns a suggested client-side name for the download data, or
         * null if there is no suggestion.  Absence of a suggested name
         * may be interpreted as a hint that the object should be
         * displayed by the user agent directly.
         */
        public String getSuggestedName();
}
```

2.1.2.2 Puzzle Framework interfaces

The component will consume and produce puzzle data in a form compatible with the Puzzle Framework 1.0 component. Although it may use the Puzzle Framework's implementations, it needs to work with these interfaces from that component:

```
package com.topcoder.util.puzzle;

/**
 * An interface defining the behavior of puzzle data, which is a named
 * collection of named attributes and resources.  Attributes are simply
 * name / value pairs, whereas resources are named blocks of binary
 * data bearing appropriate MIME media types that.  PuzzleData
 * implementations or usage environments may restrict attribute names,
 * attribute values, and resource names to be relatively short strings,
 * but details may vary.  Users are advised to keep these strings to
 * less than 256 characters, though that is not guaranteed to be a
 * sufficiently restrictive upper bound for every environment.
 */
public interface PuzzleData {

        /**
         * The standard attribute name used to store a puzzle type's name
         * among a puzzle's attributes
         */
        public static final String PUZZLE_TYPE_ATTRIBUTE;

        /**
         * Gets the name of the puzzle represented by this data
         *
         * @return
         */
        public String getName();

        /**
         * Returns the value of the named attribute of this puzzle data,
         * or null if this data has no attribute of the specified name
         *
         * @param name
         * @return
         */
        public String getAttribute(String name);
```

```
        /**
         * Returns all the attributes of this puzzle data in the form of
         * an unmodifiable Map from attribute names to attribute values
         *
         * @return
         */
        public java.util.Map getAllAttributes();

        /**
         * Returns the named resource of this puzzle data, or null if
         * this data has no resource of the specified name
         *
         * @param name
         * @return
         */
        public PuzzleResource getResource(String name);

        /**
         * Returns all theresources of this puzzle data in the form of an
         * unmodifiable Map from resource names to PuzzleResource objects
         *
         * @return
         */
        public java.util.Map getAllResources();
}


/**
 * A named and typed block of data associated with a puzzle, such as an
 * image, an audio clip, or a lengthy piece of text
 */
public interface PuzzleResource {

        /**
         * Retrieves the name of this puzzle resource as a String
         *
         * @return
         */
        public String getName();

        /**
         * Retrieves the MIME media type of this resource as a String
         *
         * @return
         */
        public String getMediaType();

        /**
         * Retrieves the data content of this resource as a byte[]
         *
         * @return
         */
        public byte[] getData();
```

```
}
```

2.1.3 Environment Requirements

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

2.1.4 Package Structure

com.orpheus.game.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:

Front Controller 2.1

Random String Image 1.0

Puzzle 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.