Administration Logic Requirements Specification

1. Scope

1.1 Overview

The Administration Logic component provides business logic in support of user manipulation tasks performed by the Orpheus application. For the most part this involves providing `Handler` and `Result` implementations conformant to the specifications of the Front Controller component version 2.1.

1.2 Logic Requirements

1.2.1 General Strategy

The Orpheus server application is designed around use of the Front Controller for business logic, JSPs for view implementations, and, for user information, the User Profile Manager for persistence. Most, if not all of the logic will be provided by `Handler` implementations, which can be strung together into chains and can direct to views (“`Results`”) as appropriate for specific request URIs and HTTP methods through the Front Controller’s configuration. Where one handler must provide data for another or for a result, the most straightforward way is to attach the data to the provided “`ActionContext`“ as a named attribute; such data could also be attached to the session or the `ServletRequest`, but the action context encapsulates it better if it does not need to be visible across multiple requests or other than to interested handlers and results.

General (as opposed to per-user) messaging is performed by application in mail drop fashion, meaning that the application records general messages on the server, from which clients periodically retrieve them.

1.2.2 Administrative Summary Handler

The component will provide a handler for extracting summary data from the application’s persistent sources and assigning it as a (configurably named) attribute(s) of the request for use by a view generator. The summary data provided in this version is the number of games in progress, the number of sponsors awaiting approval, and the number of games having pending winners awaiting approval.

1.2.3 Pending Sponsor Handler

The component will provide a handler that loads overview data about all sponsors pending approval and assigns it to request attributes of configurable name for use by a view generator. The data to be loaded are

- The user profile of each pending sponsor
- The domain objects associated with each pending sponsor, including individual approval status

1.2.4 Pending Sponsor Domain Handler

The component will provide a handler that loads detail data about a particular domain that is pending approval and assigns it to request attributes for use by a view generator. It will accept the ID of the domain to load as a request parameter (of configurable name). The data to be loaded are:

- The domain object for the specified domain
- The user profile of the domain’s associated sponsor
- The image details for each associated image, including individual approval status

1.2.5 *Sponsor Approval / Rejection Handlers*

The component will provide distinct handlers for approving and rejecting a particular pending sponsor. The sponsor's user ID will be specified to each handler via a request parameter of configurable name. The handlers will verify that the ID is associated with a sponsor user profile that has neither been approved nor rejected; if that is not so then they will produce configurable result codes and stop. Otherwise, they set the sponsor profile's IS_APPROVED property to "Y" or "N" as appropriate, then update the profile through the User Profile Manager.

1.2.6 *Domain Approval / Rejection Handlers*

The component will provide distinct handlers for approving and rejecting a particular domain associated with a pending sponsor. The sponsor's user ID and the domain ID will be specified to the handlers via request parameters having configurable names. The handlers will verify that the ID is associated with a sponsor user profile that has neither been approved nor rejected; if that is not so then they will produce configurable result codes and stop. They will furthermore verify that the specified domain is assigned to the specified sponsor, else they will likewise produce a configurable error result and stop. Otherwise, they mark the domain approved or rejected as appropriate, and records the change.

1.2.7 *Sponsor Image Approval / Rejection Handlers*

The component will provide distinct handlers for approving and rejecting a particular sponsor image. The sponsor's user ID, the domain ID, and the image ID will be specified to the handlers via request parameters having configurable names. The handlers will verify that the ID is associated with a sponsor user profile that has neither been approved nor rejected; if that is not so then it will produce configurable result codes and stop. They will furthermore verify that the specified domain is assigned to the specified sponsor and that the image is associated with the domain, else they will likewise produce configurable error results and stop. Otherwise, they mark the image approved or rejected as appropriate, and record the change.

1.2.8 *Pending Winner Handler*

The component will provide a handler that loads overview data about all games with winners pending and assigns the information to request attributes (of configurable name) for use by a view generator. The data will be provided in the form of an array of `PendingWinner` objects.

1.2.9 *Pending Winner Approval / Rejection Handlers*

The component will provide distinct handlers for approving and rejecting particular pending winners. The game ID and user ID will be specified to the handlers via request parameters having configurable names. The handlers will verify that the specified user is currently the first pending winner for the specified game; if that is not so then they will produce configurable result codes and stop. Otherwise, they record the decision.

1.2.10 *Game Parameter Handler*

The component will provide a handler that accepts general information about a game to be created, and records it in the user's session in configurably-named attributes. Information will be parsed from HTTP request parameters, whose names will also be configurable.

The following information to be collected:

- A ball color ID (type `Long`)
- Month, day, and time of day for game start; the nearest future date matching those criteria will be stored in the session (type `java.util.Date`)
- A key count (type `Integer`)

- A block count (type `Integer`)

1.2.11 Block Parameter Handler

The component will provide a handler that accepts information about one or more 'blocks' of sites in a game, and records it in the `ActionContext` in configurably-named attributes. Information will be parsed from HTTP request parameters, whose names will also be configurable.

For each block, the following information will be collected:

- A maximum time per 'slot' (type `Integer`)
- A slot count (type `Integer`)
- An auction start time (as for requirement 1.2.10, bullet 2; type `java.util.Date`)
- An auction end time (as for requirement 1.2.10, bullet 2; type `java.util.Date`)

1.2.12 Create Game Handler

The component will provide a handler that creates a new game object corresponding to previously collected parameters (per sections 1.2.10 and 1.2.11), records it via the GameData EJB, and initiates auctions for the configured blocks via an Auction Manager instance (see Auction Framework 1.0 documentation) obtained from the application context as an attribute of configurable name.

At designer option, a single handler may be provided in support of requirements 1.2.11 and 1.2.12.

1.2.13 Generate Minihunt Targets for a Domain

The component will provide a mechanism for generating 'minihunt targets' for a hosting slot. These are pairs comprising the URL of an HTML page in the domain on one hand, and the text content of some HTML element on that page on the other hand.

- The component will use the Web Site Statistics component to gather candidates for the targets from the domain associated with the slot. Minimum, maximum, and preferred target length will be provided as component configuration parameters.
- The component will base its selections in part on the number of distinct pages on which candidates appear. The minimum, maximum, and preferred number of pages will be configuration parameters.
- The component will prefer collections of targets that appear on different pages from each other.
- The component will record the targets for the slot by recording updated slot information with the GameData EJB

1.2.14 Regenerate Brainteaser Handler

The component will provide a handler that regenerates the brainteaser for a specified hosting slot. It will accept, as a configurably-named request parameter, the ID of the slot for which to regenerate the brainteaser. If the slot has already started hosting (and perhaps even finished), then the handler returns configurable result string and stops. Otherwise it (re)generates the slot's brainteasers:

- The text of the first minihunt target (see requirement 1.2.13) for the slot will be used as the base for the brainteasers; it is obtained via the GameData EJB
- The component selects the type of brain teaser at random from among the 'missing letter' and 'letter scramble' types

- The component obtains a `PuzzleTypeSource` instance from the application context as an attribute of configurable name, and uses it to obtain a `PuzzleType` object of the chosen kind (refer to the Puzzle Framework documentation for more information); the actual puzzle type names to use will be configured on this component to match their configuration on the puzzle type source
- The component obtains a puzzle generator from the PuzzleType, and uses it to create a puzzle series (number of puzzles in the series will be a configuration parameter); the component will assume that the generator requires attributes as specified for the generators provided by the Missing Letter Puzzle 1.0 and Letter Scramble Puzzle 1.0 components.
- The component records the resulting puzzles as the slot's brainteasers by first storing the puzzles themselves by use of the `AdminData` EJB, then updating the slot information via the `GameData` EJB.

1.2.15 Regenerate Puzzle Handler

The component will provide a handler that regenerates the puzzle for a specified hosting slot. It will accept, as a configurably-named request parameter, the ID of the slot for which to regenerate the brainteaser. If the slot has already started hosting (and perhaps even finished), then the handler returns configurable result string and stops. Otherwise it regenerates the slot's puzzle:

- The component obtains the image from which to create the puzzle by means of the `GameData` EJB (in the form of the `DownloadData` for the image)
- The component selects the type of puzzle at random from among the 'jigsaw puzzle' and 'sliding tile' types
- The component obtains a `PuzzleTypeSource` instance from the application context as an attribute of configurable name, and use it to obtain a `PuzzleType` object of the chosen kind (refer to the Puzzle Framework documentation for more information); the actual puzzle type names to use will be configured on this component to match their configuration on the puzzle type source
- The component obtains a puzzle generator from the `PuzzleType`, and uses it to create the puzzle (the puzzle dimensions, in pieces, will be configuration parameters on a per-puzzle type basis); the component will assume that the generator requires attributes as specified for the generators provided by the Jigsaw Puzzle 1.0 and Sliding Tile Puzzle 1.0 components.
- The component records the result as the slot's puzzle by first storing the puzzle itself by use of the `AdminData` EJB, then updating the slot information via the `GameData` EJB.

1.2.16 Reorder Slots Handler

The component will provide a Handler that reorders as-yet unreached slots within the same block by moving one slot to a different position. The handler will receive a game ID, a slot ID, and an offset via request parameters of configurable name, and will move the slot within its block by the specified offset. In no event will a slot be moved before a site that has already hosted or is currently hosting, nor will one ever be moved past the last position in its block.

1.2.17 Delete Slot Handler

The component will provide a Handler that permanently removes one as-yet unreached slot from among the future slots. It will accept a game ID and slot ID via request parameters of configurable name, and will delete the specified slot from the game.

1.2.18 Thread Safety

The component will be thread safe.

1.3 Required Algorithms

None

1.4 Example of the Software Usage

The component will be used to handle the administrative tasks supported by the Orpheus application.

1.5 Future Component Direction

Additional handlers and supporting classes will be added as changes to the application require.

2. Interface Requirements

2.1.1 Graphical User Interface Requirements

None

2.1.2 External Interfaces

2.1.2.1 `Handlers`

The component provides implementations of the Front Controller's `Handler` interface; see that component's documentation for details.

2.1.2.2 `AdministrationManager` service

The component will provide the following class, one or more instances of which will be made accessible to other components in the application server. Constructors and methods may be added to this class, but not removed. Additional unchecked exceptions may be defined for the existing methods at designer discretion.

```
package com.orpheus.administration;

/**
 * A class providing an API for administrative functions that need to
 * be accessible to other components or that need to be accessed by
 * multiple elements of this component.
 */
public class AdministrationManager {

      /**
       * (Re)generates the game-win puzzle for the specified hosting
       * slot
       *
       * @param slotId
       * @throws AdministrationException if a checked exception
       *     prevents this method from completing normally
       */
      public void regeneratePuzzle(long slotId)
            throws AdministrationException {
      }

      /**
       * (Re)generates the brain teaser for the specified hosting slot
```

```
         *
         * @param slotId
         * @throws AdministrationException if a checked exception
         *    prevents this method from completing normally
         */
        public void regenerateBrainTeaser(long slotId)
              throws AdministrationException {
        }

        /**
         * (Re)generates the mini-hunt targets for the specified hosting
         * slot
         *
         * @param slotId
         * @throws AdministrationException if a checked exception
         *    prevents this method from completing normally
         */
        public void generateHuntTargets(long slotId)
              throws AdministrationException {
        }

        /**
         * Initializes all the slots in the specified hosting block by
         * generating minihunt targets, brain teasers, and game-win
         * puzzles for them.
         *
         * @param blockId
         * @throws AdministrationException if a checked exception
         * prevents this method from completing normally
         */
        public void initializeSlotsForBlock(long blockId)
              throws AdministrationException {
        }
  }

/**
 * An exception indicating an unrecoverable failure in performing
 * administrative functions or recording or accessing administrative
 * data
 */
public class AdministrationException
      extends com.topcoder.util.errorhandling.BaseException {
}
```

2.1.2.3 `AdminData` Component Interface

The AdminData EJB presents this component interface

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

/**
 * 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;
```

```
    /**
     * Approves the specified sponsor
     *
     * @param sponsorId
     * @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 approveSponsor(long sponsorId)
          throws java.rmi.RemoteException,
          com.orpheus.administration.AdministrationException;

    /**
     * Rejects the specified sponsor
     *
     * @param sponsorId
     * @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 rejectSponsor(long sponsorId)
          throws java.rmi.RemoteException,
          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 {

      /**
       * Returns the user ID of the pending winner
       *
       * @return
       */
      public long getPlayerId();

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

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

2.1.2.4 `GameData` Component Interface

The component makes use of the `GameData` EJB provided by the Orpheus Game Peristence component, which exposes these interfaces:

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

/**
 * The 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 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;

      /**
       * Retrieves a Game object representing the 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;

/**
 * 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;

/**
 * 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 domain
 * @return
 */
public Game[] findGamesByDomain(String domain, long playerId)
      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
 *
 * @param playerId
 * @param slotId
 * @param date
 */
public void recordSlotCompletion(long playerId, long slotId,
      java.util.Date date) 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 hosting slots completed by any player in the
 * specified game, and returns the results as an array of
 * HostingSlot objects
 *
 * @param gameId
 * @return
 */
public HostingSlot[] findCompletedSlots(long gameId)
      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;
```

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

/**
 * Updates the persistent hosting slot information for the
 * existing slots represented by the specified HostingSlot
 * objects to reflect the current state of those objects.
 * (Objects are matched to persistent data via their IDs.)
 *
 * @param slots
 */
public void updateSlots(HostingSlot[] slots)
      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;

/**
 * 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;

/**
 * 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;
      /**
       * 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;
}

/**
 * The Game interface represents a BarookaBall game.  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 the ball image corresponding to this
         * BallColor.
         *
         * @return
         */
        public long getImageId();
  }
```

```
/**
 * Represents a "block" of Barooka Ball hosts.  Blocks serve as an
 * organizational unit for Ball 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();

      /**
       * Sets the hosting slots for this block
       *
       * @param slots
       */
      public void setSlots(HostingSlot[] slots);

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

/**
 * An interface representing a hosting domain within the Orpheus
 * 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 yet 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, or null if none has
         * yet been assigned
         *
         * @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();
    }

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

        /**
         * Returns the unique identifier of this 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();
    }

    /**
     * Represents an object to be found on an Orpheus 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();
    }
```

2.1.3 Environment Requirements

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

2.1.4 Package Structure

com.orpheus.administration

3. Software Requirements

3.1 Administration Requirements

3.1.1 What elements of the application need to be configurable?

Various request parameter names and result strings as described among the logic requirements.

JNDI name of the AdminData EJB's home interface.

JNDI name of the GameData EJB's home interface.

3.2 Technical Constraints

3.2.1 Are there particular frameworks or standards that are required?

Java Servlet API 2.4

3.2.2 TopCoder Software Component Dependencies:

Front Controller 2.1

User Profile 1.0

User Profile Manager 1.0

Web Site Statistics 1.0

Puzzle Framework 1.0

Auction Framework 1.0

Orpheus Administration Persistence 1.0

Orpheus Game Persistence 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

3.4.2 Help / User Documentation

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