Game Operation Logic Requirements Specification

1. Scope

1.1 Overview

The Orpheus Game Logic components provide business logic in support of game play and game data manipulation tasks performed by the Orpheus application. For the most part this involves providing `Handler` implementations conformant to the specifications of the Front Controller component version 2.1. This game logic component focuses generally on handling user actions that require updating persistent application data, or that involve non-trivial computation or data processing within handler or result objects.

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, the User Profile Manager for user access and manipulation, and EJBs for most persistence purposes. Most 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 it 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 other than to interested handlers and results or across multiple requests.

1.2.2 Handlers

The component will provide `Handler` implementations for servicing HTTP requests related to game play activities supported by the application.

1.2.2.1 Register for Game Handler

The component will provide a `Handler` that registers the current (logged-in) user for a specified game. The game to register for will by identified by its unique ID, parsed from a request parameter of configurable name.

1.2.2.2 Message Poll Result

The component will provide a `Result` for responding to message polling requests. It will obtain a `DataStore` from an application context attribute of configurable name and use it to construct an Atom 1.0 feed document, which it returns (in XML format) as the HTTP response entity. It will select items for the feed as follows:

- The `Result` will be configured with a request parameter name with which it will accept an item update date and time. If the request specifies a parameter of that name then this `Result` will parse it as a `java.util.Date` in ISO 8601:2004 format (http://en.wikipedia.org/wiki/ISO_8601), and only items with a creation or update date strictly later than the parsed date will be included in the resulting feed.
- The `Result` will determine the games for which the requesting player (the one currently logged-in) is registered; the name of each will be used as an item category, and items in any of those categories will be included in the feed, subject to the date restriction already described.
- The `Result` will be configured with zero or more category names for categories whose items are always included in the feed, subject to the date restriction criterion.

The response will be assigned the content type registered for Atom, “application/atom+xml”.

1.2.2.3 Key Submission Handler

The component will provide a Handler that processes key submissions. It will accept a game ID via a request parameter of configurable name, and several key strings as multiple values of another request parameter, also of configurable name. It will determine whether the submitted keys match those recorded for the currently logged-in player for the specified game for the most recently completed (by any player) *k* hosting slots, where *k* is the key count of the specified game.

The handler will maintain a count, in the player’s session, of the number of failures to submit matching keys for the specified game since the last time its hosting slot changed. (The ID of the most recently-completed hosting slot in that game is one possible measure of whether the hosting slot has changed.)

Contingencies and results:

- If the specified game is not active then the handler returns a configurable result code characteristic of the problem, regardless of the values of the keys
- If the keys do not match then the player’s failure count for the game and slot is incremented. The handler then returns a configurable result string characteristic of whether or not the failure count meets or exceeds a configurable threshold value.
- If the keys do match then the handler records a game completion for the player in the specified game (via the game data persistence component), then returns null

1.2.2.4 Application Messaging Handler

The component will provide a handler that supports general-purpose messaging between the application and users via the Messenger Framework. It is configured with the name of a `MessengerPlugin` and with one or more mappings between message property names and request parameter names or request / session / application / action context attribute names. The handler obtains a `MessengerPlugin` instance, creates a `MessageAPI` instance, assigns values to the message parameters as directed by its configuration, then dispatches the message via the `MessengerPlugin`.

1.2.2.5 Puzzle Rendering Handler

The component will provide a `Handler` that produces a rendition of a puzzle in `String` form and attaches it to the request as an attribute of configurable name. The handler will be configured also with the name of a request attribute from which to obtain the ID of the puzzle to render (as a `Long`), the media type in which to render it (a `String`), the name of the application context attribute from which to obtain a `PuzzleTypeSource`, and the base name of a session attribute to which to assign a `SolutionTester`. The handler operates in this fashion:

- It obtains from game data persistence a `PuzzleData` for the specified ID.
- It extracts the puzzle type name from among the `PuzzleData`’s attributes, and feeds it to a `PuzzleTypeSource` to obtain the appropriate `PuzzleType`.
- It obtains a renderer for the specified medium from the PuzzleType, and uses it to render the puzzle.
- It assigns the `SolutionTester` resulting from the rendering to a session attribute named by concatenating the configured base attribute name with the puzzle ID.
- It assigns the rendered puzzle, as a String, as a request attribute with the configured name.

1.2.2.6 Puzzle Solution Handler

The component will provide a `Handler` that tests whether the HTTP request parameters describe a solution to a previously presented puzzle. It determines the IDs of the game and slot to which the puzzle applies from request parameters of configurable name. It then finds the appropriate solution tester:

- it looks up the ID of the slot's puzzle among the slot information,
- appends that puzzle ID to a configured prefix to obtain the name of the session attribute containing the appropriate `SolutionTester`, and
- retrieves the tester from the session via the computed attribute name

The handler will pass the request's full parameter map to the tester. If the solution is correct then the handler records a game completion for the currently logged-in player in the game whose ID is available from a request parameter of configurable name, and the handler returns null.

If the solution is incorrect then the game is forced to advance to the next host, and a slot completion is recorded for the current player in the specified slot, the `SlotCompletion` object is assigned to a request attribute of configurable name, and the handler returns a configurable result code string.

1.2.2.7 Winner Data Handler

The component will provide a `Handler` that records contact information collected from players when they win the game. Information will be stored in the winner's User Profile, and recorded using the User Profile Manager. The handler's configuration will specify profile types that the handler must ensure are present in the profile, and will specify a map from parameter names to corresponding user profile property names. The Handler will install the profile type(s) if necessary, then copy the parameter values to the profile and store it.

1.2.2.8 Plug-in Download Handler

The component will provide a `Handler` that records an instance of a plug-in download. It will use a plug-in name taken from a request parameter of configurable name to identify the plug-in downloaded to the game data persistence component.

1.2.3 EJB Usage

The component will rely for persistence on an EJB and various data transfer objects provided by a corresponding persistence component (see 2.1.2.2 and 2.1.2.3). It will have the ability to access the bean via its remote interfaces. It is desired that the component also support usage of the bean's local interfaces instead when they are available.

1.3 Required Algorithms

None

1.4 Example of the Software Usage

The component will be used to handle the game management tasks exposed by the application's external interface.

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

The component provides implementations of the Front Controller's `Handler` interface and uses various functionalities of the Auction Framework; see those components' documentation for details.

2.1.2.1 Determining the logged-in user

The component will rely on the following class and method to obtain information for the currently logged-in user at need:

```
package com.topcoder.web.user;

public class LoginHandler implements Handler {

      /**
       * Returns a UserProfile representing the user, if any, that has
       * successfully logged in in the context of the specified HTTP
       * session.
       */
      public static UserProfile getAuthenticatedUser(
            HttpSession session) {
      }
}
```

2.1.2.2 Data Transfer Objects

The component will use the following data transfer object interfaces provided by its corresponding persistence component:

```
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();
    }
```

2.1.2.3 EJB Interfaces

The component will use the following EJB home and component interfaces provides by its corresponding persistence component:

```
    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;
}
```

2.1.3 Environment Requirements

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

2.1.4 Package Structure

com.orpheus.game

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.

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

Orpheus Game Persistence 1.0

User Profile 1.0

User Profile Manager 1.0

Site Validator 1.0

Messenger Framework 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

3.4.2 Help / User Documentation

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