Auction Logic Requirements Specification

1. Scope

1.1 Overview

The Auction Logic component provides business logic in support of auction management 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.

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 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 Open Auctions Handler

The component will provide a `Handler` that retrieves the currently open auctions (those that have started but not yet ended) and assigns them (in the form of an `Auction[]`) as a request attribute for use in generating a view. The name of the request attribute will be configurable.

1.2.3 Leading Bids Handler

The component will provide a `Handler` that determines the current leading bids for a specified auction and assigns them (in the form of a `Bid[]`) as a request attribute for use in generating a view. The ID of the relevant auction will be parsed from a request parameter of configurable name, and the name of the request attribute to which the bid array is assigned will likewise be configurable.

1.2.4 Bid Placement Handler

The component will provide a `Handler` implementation that provides for placing new bids in an open auction. To create a `Bid` object corresponding to this bid the handler will use the ID of the currently logged-in user as the bidder ID, will assign a timestamp corresponding to the current time, and will parse an auction ID, image ID, and maximum amount from request parameters of configurable name.

1.2.5 Bid Update Handler

The component will provide a `Handler` implementation that updates existing bids in an open auction. To find the original bid the handler will parse an auction ID and bid ID from request parameters of configurable name, will retrieve the corresponding `Auction`, and will search the `Bids` to find the one with the correct ID. To create a `Bid` object for the updated bid the handler will use the ID of the currently logged-in user as the bidder ID, will parse a maximum amount from a request parameters of configurable name, and will copy all other parameters from the original bid.

1.2.6 Bid Validator

The component will provide a `BidValidator` implementation that enforces the following rules:

- New bids and bid updates are invalid if the specified auction has not yet started or has already completed.
- New bids are invalid if their maximum amount is not at least as great as the auction's minimum bid.
- Bid updates are invalid if their maximum amount is not greater than that of the original bid, or if the IDs, bidder IDs, or image IDs differ.
- Bid updates are invalid if the specified original bid is not among the bids in the specified auction.

1.2.7 Auction Listener

The component will provide an `AuctionListener` that responds to auction completion events by notifying the game data manager service of the IDs of the winning bids and then requesting that the administration manager service initialize the slots for the corresponding block. For this purpose, the component will assume that the block ID required by the game data manager is the same as the auction ID of the completed auction.

1.3 Required Algorithms

None

1.4 Example of the Software Usage

The component will be used to handle the auction 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 Recording Winning Bids

The component will use a GameDataManager instance obtained via an application context attribute (of configurable name) to record winning bids. The component will assume that auctions and associated hosting blocks share IDs. A partial description of this interface is below.

```
package com.orpheus.game;

/**
 * Defines the behavior of objects that manage Orpheus game data
 */
public interface GameDataManager {

      /*
       * Records the IDs of the winning bids for the slots in the
       * specified hosting block
       *
       * @param blockId
       * @param bidIds
       * @throws GameDataException if a checked exception prevents this
       *    method from completing successfully, or if bid IDs have
       *    already been assigned to the block
       */
      public void recordWinningBids(long blockId, long[] bidIds)
            throws GameDataException;
}

/*
 * An exception indicating a failure to modify game data
 */
public class GameDataException
      extends com.topcoder.util.errorhandling.BaseException {
}
```

The component will rely on Bids being of this implementation type:

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

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

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

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

2.1.2.3 Initializing Slots

The component will use an `AdministrationManager` instance obtained via an application context attribute (of configurable name) to initialize slots. The component will assume that auctions and associated hosting blocks share IDs. A partial description of this interface is below.

```
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 {

      /**
       * 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 {
}
```

2.1.3 Environment Requirements

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

2.1.4 Package Structure

com.orpheus.auction

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

Auction Framework 1.0

Orpheus Auction Persistence 1.0

Orpheus Game Logic 1.0

User Profile 1.0

User Profile Manager 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.