# Auction Logic Component Specification

## 1. Design

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.

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.

### 1.1 Design Patterns

Observer Pattern is used by AuctionListenerImpl and AuctionManager from the Auction Framework component to deal with the action events.

### 1.2 Industry Standards

Java Servlet API 2.4

### 1.3 Required Algorithms

The xml Element passed to the handler's constructor must follow the DTD defined below, however developers can choose to validate the xml or not. (We can assume the xml element is valid, and throw IAE if failed to extract what we want).
The xml element structure is relatively simple, please use DOM API to extract node values.

#### *1.3.1 OpenAuctionsHandler*

The DTD of the xml element:

```
<!ELEMENT handler (open_auction_key)>
<!ATTLIST handler type CDATA #REQUIRED>
<!ELEMENT open_auction_key (#PCDATA)>
```

Follow is a sample xml:

```
<handler type="OpenAuctions">
   <open_aution_key>
      open_auction_key
   </open_aution_key >
</handler>
```

Following is simple explanation of the above XML structure.
The handler's type attribute is required by Front Controller component, it won't be used in

this design. It will be referred by the global configuration file in 4.3.

The open_auction_key node's value represents the http request attribute key to store the auctions.

This handler 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.

### *1.3.2 LeadingBidsHandler*

The DTD of the xml element:

```
<!ELEMENT handler (action_id_param_key, leading_bids_key)>
<!ATTLIST handler type CDATA #REQUIRED>
<!ELEMENT auction_id_param_key (#PCDATA)>
<!ELEMENT leading_bids_key (#PCDATA)>
```

Follow is a sample xml:

```
<handler type="LeadingBidsHandler">
   <auction_id_param_key>
      auction_id_param_key
   </auction_id_param_key>
   <leading_bids_key>
      Leading_bids_key
   </leading_bids_key>
</handler>
```

Following is simple explanation of the above XML structure.

The handler's type attribute is required by Front Controller component, it won't be used in this design. It will be referred by the global configuration file in 4.3.

The action_id_param_key's node value represents the http request parameter key to get the auction id parameter.

The leading_bids_key's node value represents the http request attribute key to store the leading bids.

This handler 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.

### *1.3.3 BidPlacementHandler*

The DTD of the xml element:

```
<!ELEMENT handler (action_id_param_key,
   max_amount_param_key, image_id_param_key)>
<!ATTLIST handler type CDATA #REQUIRED>
<!ELEMENT auction_id_param_key (#PCDATA)>
<!ELEMENT max_amount_param_key (#PCDATA)>
<!ELEMENT image_id_param_key (#PCDATA)>
```

Follow is a sample xml:

```
<handler type="BidPlacementHandler">
   <action_id_param_key>
      action_id_param_key
   </action_id_param_key>
   <max_amount_param_key>
      max_amount_param_key
   </max_amount_param_key>
   <image_id_param_key>
      image_id_param_key
   </image_id_param_key>
</handler>
```

Following is simple explanation of the above XML structure.

The handler's type attribute is required by Front Controller component, it won't be used in this design. It will be referred by the global configuration file in 4.3.

The action_id_param_key's node value represents the http request parameter key to get the auction id parameter.

The max_amount_param_key's node value represents the http request parameter key to get the max amount id parameter.

The image_id_param_key's node value represents the http request parameter key to get the image id parameter.

This handler 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.

### *1.3.4 BidUpdateHandler*

The DTD of the xml element:

```
<!ELEMENT handler (action_id_param_key,
   max_amount_param_key, bid_id_param_key)>
<!ATTLIST handler type CDATA #REQUIRED>
<!ELEMENT auction_id_param_key (#PCDATA)>
<!ELEMENT max_amount_param_key (#PCDATA)>
<!ELEMENT bid_id_param_key (#PCDATA)>
```

Follow is a sample xml:

```
<handler type="x">
   <action_id_param_key>
      action_id_param_key
   </action_id_param_key>
   <max_amount_param_key>
      max_amount_param_key
   </max_amount_param_key>
   < bid_id_param_key>
      bid_id_param_key
```

```
    </ bid_id_param_key>
</handler>
```

Following is simple explanation of the above XML structure.

The handler's type attribute is required by Front Controller component, it won't be used in this design. It will be referred by the global configuration file in 4.3.

The action_id_param_key's node value represents the http request parameter key to get the auction id parameter.

The max_amount_param_key's node value represents the http request parameter key to get the max amount id parameter.

The bid_id_param_key's node value represents the http request parameter key to get the bid id parameter.

This handler 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 `Bid`s 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.

## 1.4 Component Class Overview

**BidUpdateHandler**:

BidUpdateHandler 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 `Bid`s 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.

This class is thread safe since it does not contain any mutable state.

**BidPlacementHandler**:

BidPlacementHandler 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.

This class is thread safe since it does not contain any mutable state.

**LeadingBidsHandler**

LeadingBidsHandler 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.

This class is thread safe since it does not contain any mutable state.

**OpenAuctionHandler**:

OpenauctionHandler 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.

This class is thread safe since it does not contain any mutable state.

**KeyConstants**:

The KeyConstants class defined three key constants used to get the AuctionManager, GameDataManager and AdministratorManager from the ServletContext attributes..

**BidValidatorImpl**:

The BidValidatorImpl is used by AuctionManager to validate the bid before creating a new bid or updating a bid.

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.

This class is thread safe since it's immutable.

**AuctionListenerImpl**:

An AuctionListener implementation 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 listener will assume that the block ID required by the game data manager is the same as the auction ID of the completed auction.

This class is thread safe since it does not contain any mutable state.

## 1.5 Component Exception Definitions

**IllegalArgumentException**

This exception is thrown in various methods if null object is not allowed, or the given string argument is empty. Refer to the documentation in Poseidon for more details.

**NOTE: Empty string means string of zero length or string full of whitespaces.**

**HandlerExecutionException**

It is thrown from OpenAuctionsHandler, LeadingBidsHandler, BidPlacementHandler and

BidUpdateHandler classes..

### 1.6 Thread Safety

Though this component is not completely thread safe, it could be used in a thread safe manner. All the four handler implementations are immutable and thread safe. The KeyConstants, BidValidatorImpl and AuctionListenerImpl are all immutable and thread safe. CustomBidImpl is mutable and not thread safe. However, in the current version, the instance of CustomBid will not be shared among multi-threads.

## 2. Environment Requirements

### 2.1 Environment

Java 1.4 or higher.

### 2.2 TopCoder Software Components

**Configuration Manager 2.1.5**

It is used to load configuration values.

**Front Controller 2.1**

The Handler interface comes from this component

**Auction Framework 1.0**

The Auction entities come from this component.

**Orpheus Auction Persistence 1.0**

The OrpheusBid class comes from this component.

**Orpheus Game Logic 1.0**

The GameDataManager class comes from this component.

**Web Application User Logic 1.0**

The LoginHandler class comes from this component.

**User Profile 1.0**

The UserProfile class comes from this component.

### 2.3 Third Party Components

None.

## 3. Installation and Configuration

### 3.1 Package Name

com.orpheus.auction

### 3.2 Configuration Parameters

Configuration values for com.orheus.auction.keyConstants

| Parameter | Description | Values |
|---|---|---|
| **autction_manager** | The value is used get the AuctionManager instance from the ServletContext. **Required**. | auction_manager |
| **game_data_manager** | The value is used get the GameDataManager instance from the ServletContext. **Required**. | game_data_manager |
| **administrator_manager** | The value is used get the AdministratorManager instance from the ServletContext. **Required**. | administrator_manager |

### 3.3 Dependencies Configuration

ConfigurationManager should be properly configured to make this component work.

## 4. Usage Notes

### 4.1 Required steps to test the component

- Extract the component distribution.
- Follow Dependencies Configuration.
- Execute 'ant test' within the directory that the distribution was extracted to.

### 4.2 Required steps to use the component

Preload the configuration file into Configuration Manager. Follow demo.

### 4.3 Demo

```
Assume the configuration file in 3.2 is used here. And follow the
algorithm section about how to configure the handlers.
```

#### *4.3.1 Global file for all the handlers*

```
<global>
  <actions-def>
     <!-- the default action from Front Controller component -->
    <action-def name="default">
       com.topcoder.web.Front Controller.DefaultAction
    </action-def>
  </actions-def>

  <handlers-def>
   <handler-def name="OpenAuctionsHandler">
      com.orpheus.auction.OpenAuctionsHandler
    </handler-def>
   <handler-def name="LeadingBidsHandler">
```

```
            com.orpheus.auction.LeadingBidsHandler
        </handler-def>
        <handler-def name="BidPlacementHandler">
            com.orpheus.auction.BidPacementHandler
        </handler-def>
        <handler-def name="BidUpdateHandler">
            com.orpheus.auction.BidUpdateHandler
        </handler-def>
    </handlers-def>
</global>
```

#### *4.3.2 How to configure the specified handlers*

The algorithm 1.3 lists the DTD file and sample XML files for all the handlers.

All the configuration files in algorithm section give the real data, and explains how the handlers are expected to work in the Front Controller 2.1 framework. Please refer to that section about how to configure the specified handlers.

## 5. Future Enhancements

More handler implementation could be added.