# Design: CFLEX2.0 AIM Auction Creation

# Overview

## Requirement:

* Flex User should able create an AIM Auction (Simple Auction or Strategy Auction) for an existing product as well as for new product using new Web-CAS.

## Scope:

AIM Auction and AIM AON Auction creation for FLEX\_MAIN session

# Implementation

## 2.1 IDL Changes

### 2.1.1 CMI IDL Modification

* None

### Other IDL Modification:

* None

## 3. Global Server

### 3.1 Classes & Interfaces

* None

## 3.2 Scripts

* None

## 3.3 Database Changes

* None

## 3.4 SetContext

* None

## 3.5 XML/DTD

* None

## 4. Client

### WebCAS Changes

### 4.1.1 Classes & Interfaces

### WebCasOrderEntryDispatcher.java

### We need to add two new action id for creating an AIM Auction creation on simple and Strategy product.

### WebCasOrderEntry.java

### We need to add below methods for creating an AIM auction on simple and strategy product.

### public InternalizationAuctionResultStruct acceptInternalizationOrdersForProduct(ProductStruct productStruct, OrderEntryStruct primaryOrder, OrderEntryStruct matchOrder, short matchType) throws SystemException, CommunicationException, AuthorizationException, DataValidationException, NotAcceptedException, TransactionFailedException

### public InternalizationAuctionResultStruct acceptInternalizationOrdersForStrategyProduct(ProductStruct strategyProduct,StrategyLegDefinitionStruct[] legDefinitions, OrderEntryStruct primaryOrder, LegOrderEntryStructV2[] primaryOrderLegEntries, OrderEntryStruct matchOrder,LegOrderEntryStructV2[] matchOrderLegEntries, short matchType) throws SystemException, CommunicationException, AuthorizationException, DataValidationException, NotAcceptedException, TransactionFailedException

### WebCasOrderEntryImpl.java

### We need to add implementation for the new methods added in WebCasOrderEntry for AIM auction creation.

### Sequence diagram:

### Flow for simple AIM Auction Creation:

### 

## 5. GUI

### 5.1 FlexGUI Changes

### 5.1.1 Changes for AIM Auction Creation

We need to create below mentioned files for AIM Auction creation.

1. **AIMAuctionModel.java**

This would gather all the data from GUI required for starting an AIM Auction. If product details contain only one leg, we would be creating a ProductStruct, otherwise we would be creating an ProductStruct along with an array of StrategyLegDefinitionStruct.

1. **AIMAuctionPanel.java**

This class would be having all the components required for AIM Auction creation, example, Primary order and Matched order details, etc.

1. **AuctionSelectorPanel.java**

This panel would be having the FLEX Auction panel and AIM auction panel. Based on the radio button selection, we would be showing the appropriate panel to user.

1. **AuctionServiceAPI.java**

We need to add implementation for creating an AIM auction on a simple and a strategy product.

### 5.1.2 Changes for Respond to AIM Auction

We need to create below mentioned files for AIM Auction creation.

1. **RespondToAIMAuctionFrame.java**

We would create a new frame to show the Respond to AIM auction pop-up screen.

1. **RespondToAIMAuctionPanel.java**

This class would be render all the components required to respond to AIM Auction creation, Product details, price and quantity, clearing information for that auction response.

### 5.1.3 Changes for All Event Screen

In All events screen, we would be using the last sale events to show the trade information of all trades happened during the day. It would be also listening on auction events. We would be adding separate row for each auction events like New Flex auction, T1 Ended, T2 ended.

We would be also adding the order events of logged-in user in this table. User won’t be able to see order events of other user.

We need to create below mentioned files to show my trades information.

1. **AllEventMessage.java**

Define an interface which would define common methods for all the events.

1. **AllEventMessageImpl.java**

We need to define the implementation for all events messages.

1. **AllEventsPanel.java**

We need to create a panel which would render the all events table.

1. **AllEventTableConfig.java**

We need to create a table configuration for all events table.

1. **AllEventTableModel.java**

We would define a table model which would always append the events received on GUI.

1. **AllEventTableRow.java**

We need to define the column headers and the initial ordering of these columns in the table.**5.1.4 Changes for Trades and My Trade**

In Trades screen, we would be using the last sale events to show the trade information of all trades happened during the day.

In My Trades screen, we would be listening on order events. If an order is busted or filled, we would be adding the filled and bust event in my trades screen.

We need to create below mentioned files to show my trades information.

1. **BustReportCache.java**

We would be creating a cache to store all the bust reports for the logged in user.

1. **FilledReportCache.java**

We would be creating a cache to cache all the fill reports for the logged in user.

1. **MyTradePanel.java**

We would create a new panel to render my trade table. Whenever user double clicks on any of the trade, we would be showing him the fill report screen with order and fill information. He can print or save this Jasper report.

1. **MyTradesTableModel.java**

We need to create a table model which has all the fill reports.

1. **MyTradesTableRow.java**

We need to define the set of columns headers and which column can be sorted

.

1. **ReportStructWrapper.java**

We need to create a wrapper which provides all required fields for my trade and trades screen.

1. **TradesPanel.java**

We would create a new panel to render trades table. This would be view only information. We won’t be providing any action on this table.

**5.1.5 Changes for Current Market**

We need to create below mentioned files to consume current market on Flex-GUI.

1. **CurrentMarketEventListener.java**

We need to write a new CometD listener which would continuously listen on current market events. Whenever it receives an event for current market update, it will publish this event on IEC channel.

1. **CurrentMarketCache.java**

We would be creating a cache which would subscribe on current market channel of IEC for updates. Whenever CurrentMarketEventListener publish a current market event, it will update its cache for the particular product.

1. **ProductInfoTableModel.java**

The product information panel would be showing the current market for selected products. So we this model would also subscribe for current market updates on IEC channel. If CurrentMarketEventListener publishes a current market event for any of these products, we would be updating the current market value for received product.

### 5.1.6 Sequence Diagram:

* 1. **AIM Auction Flow:**

#### 

* 1. **AIM Auction Response Flow:**

