



CBOE Application Programming Interface

Version 9.0.2

CBOE Application Server Volume 2: CAS Simulator for Stand Alone Testing

Programmer's Guide for the CBOE Application Server

CBOE PROPRIETARY INFORMATION

15 July 2011

Document #[CAS-02]

Front Matter

Disclaimer

Copyright © 1999-2011 by the Chicago Board Options Exchange (CBOE), as an unpublished work. The information contained in this document constitutes confidential and/or trade secret information belonging to CBOE. This document is made available to CBOE members, member firms and other appropriate parties to enable them to develop software applications using the CBOE Market Interface (CMi), and its use is subject to the terms and conditions of a Software License Agreement that governs its use. This document is provided “AS IS” with all faults and without warranty of any kind, either express or implied.

Table of Contents

FRONT MATTER	I
DISCLAIMER	I
TABLE OF CONTENTS	2
CHANGE NOTICES	1
ABOUT THIS DOCUMENT	3
PURPOSE	3
INTENDED AUDIENCE	3
PREREQUISITES	3
RELATED DOCUMENTS	3
SUPPORT AND QUESTIONS REGARDING THIS DOCUMENT	4
INTRODUCTION	5
INTRODUCTION TO THE CAS SIMULATOR	5
GETTING STARTED	5
INSTALLING THE CAS SIMULATOR	5
CAS SIMULATOR OPERATION	6
<i>Using the CAS Simulator</i>	7
<i>Starting Market Data</i>	7
<i>Equity Menu</i>	9
<i>Starting Log Audit</i>	12
<i>Stopping the CAS Simulator</i>	12
<i>Callback Removal</i>	13
CAS SIMULATOR RESPONSES BY INTERFACE	14
<i>Market Query Interface</i>	15
<i>Order Entry Interface</i>	18
<i>Order Query Interface</i>	24
<i>Product Definition Interface</i>	25
<i>User History Interface</i>	26
<i>Product Query Interface</i>	26
<i>Trading Session Interface</i>	27
<i>Quote Interface</i>	28
<i>IntermarketQuery Interface</i>	29
<i>IntermarketHeldOrderEntry Interface</i>	30
<i>IntermarketManualHandling Interface</i>	30
<i>NBBOAgentSessionManager Interface</i>	32
<i>NBBOAgent Interface</i>	32
<i>IntermarketUserSessionManager Interface</i>	33
<i>IntermarketUserAccess Interface</i>	33
<i>FloorTradeMaintenanceService Interface</i>	33
<i>EventGUIHelper</i>	34

Change Notices

The following change notices are provided to assist users of the CMi in determining the impact of changes to their applications.

Date	Version	Description of Change
15 Jul 2011	V9.0.2	No changes
29 Apr 2011	V9.0.1	No changes
14 Jan 2011	V9.0	No changes
08 Jan 2010	V7.0	No changes
07 Aug 2009	V6.1	No changes
22 May 2009	V6.0	New interface for FloorTradeMaintenanceService New example 17 for Market Maker Hand Held functionality
25 Nov 2008	V5.3	No changes
24 Sept 2008	V5.2	No changes
18 Jul 2008	V5.1	No changes
29 Feb 2008	V5.0	Updated the OrderEntry interface with acceptInternalizationStrategyOrder
18 Jan 2008	V4.2.4	No changes
02 Nov 2007	V4.2.3	No changes
01 June 2007	V4.2.2	No changes
23 Feb 2007	V4.2.1	No changes
15 Dec 2006	V4.2	No changes
20 Sept 2006	V4.1	No changes
25 May 2006	V4.0	New menu item for Market Data Express (MDX) New section: Callback Removal
06 Jan 2006	V3.2b	No changes
12 Aug 2005	V3.2	No changes.
29 Jul 2005	V3.2	No changes
08 Apr 2005	V3.1	No changes.
17 Dec 2004	V3.1	Internalization and auctions
18 Jun 2004	V3.0	New functionality for V3 methods
28 Apr 2004	V2.52	No changes.

Date	Version	Description of Change
06 Feb 2004	V2.63	No changes.
10 Oct 2003	V2.62	Added new screens and functionality for stock.
03 Oct 2003	V2.61	Add stock functionality
29 Aug 2003	V2.61	No changes.
31 Jul 2003	V2.6	Support for Market Linkage.
08 Jul 2003	V2.51	Revisions and additions of the V2 methods.
14 Mar 2003	V2.5	Support for Hybrid
24 Jan 2003	V2.1	Support for Linkage P orders
20 May 2002	V2.0.1	Updated Market Data information.
22 Apr 2002	V2.0	Production Release
27 Feb 2002	V2.0b	Software Development Kit Beta 2
23 Jan 2002	V2.0a	Added new methods introduced in Cmi Version 2.0. Major changes are the introduction of strategies and dynamic book depth in Version 2.0.
04 May 2001	V1.0b	Added Market Data role information.
16 Mar 2001	V1.0a	Error corrections and updated to reflect that strategies will not be part of Version 1.0.
22 Jan 2001	V1.0	Includes revisions to the CMi API since the last update. Refer to the Release Notes for full details.
22 Sep 2000	V0.9	Includes revisions to the CMi API since the last update. Refer to the Release Notes for full details.
28 Apr 2000	V0.8	Includes revisions to the CMi API since the last update. Refer to the Release Notes for full details.
30 Sep 1999	V0.5	First Publication

About This Document

Purpose

This document describes the operation of the CBOE Application Server (CAS) Simulator.

Intended Audience

Software developers, system administration personnel, and anyone using the CMi to access the CBOE markets. Also included is anyone involved in testing and certifying an application written using the CMi.

Prerequisites

This document assumes that the reader has a working knowledge of CORBA and one of the programming languages supported by CORBA, such as C++ or Java. See the reference section of a list of books and web sites that can provide you with fundamentals on CORBA. Specifically, you should be familiar with: CORBA modules, interfaces, structs, operations, IORs, exceptions, and call backs.

You should have already read API Volume 1: Overview and Concepts, API Volume 2: CMi Programmer's Guide, and CAS Volume 1: Overview and Concepts.

Related Documents

Document Number	Document Description
Roadmap.doc	CBOE API and CAS Document Road Map
API-01	CBOE API Volume 1: Overview and Concepts
API-02	CBOE API Volume 2: CMi Programmer's Guide
API-03	CBOE API Volume 3: CMi Programmer's Guide to Messages and Data Types
API-04	CBOE API Volume 4: CMi Dictionary of Attributes and Operations
API-05	CBOE API Volume 5: Using CMi with Specific Object Request Brokers
API-06	CBOE API Volume 6: Connecting to the CBOE Network
API-07	CBOE API Volume 7: CBOEdirect Certification and Testing Procedures
API-08	CBOE API Volume 8: CMi Programmer's Guide to the Market Data Express (MDX) data feed
CAS-01	CBOE Application Server Volume 1: Overview and

Document Number	Document Description
	Concepts

Support and Questions Regarding This Document

Questions regarding this document can be directed to The Chicago Board Options Exchange at 312.786.7300 or via e-mail: api@cboe.com.

The latest version of this document can be found at the CBOE web site <http://systems.cboe.com>.

Introduction

The CBOE is adding new interfaces to provide access to exchange services. These interfaces are designed to support both electronic and open outcry trading at the Chicago Board Options Exchange.

The first of these interfaces is an Application Programming Interface (API) that provides access to all exchange trading services and is targeted at firms making markets at CBOE. This API is known as the *CBOE Market Interface* (CMi). CMi is a distributed object interface based upon the CORBA (Common Object Request Broker Architecture) standard from the Object Management Group (OMG). The interface is defined using the Interface Definition Language (IDL), which is an OMG and ISO standard. Messages are transported using the Internet Inter-Orb Protocol (IIOP), which operates over standard Internet protocols (TCP/IP).

The second interface is a message-based protocol based upon the Financial Information Exchange (FIX) protocol. CBOE's implementation of the FIX protocol should be of particular interest to retail and institutional firms that require order routing to CBOE markets. The FIX protocol is implemented over TCP/IP. FIX is an important messaging protocol in the financial industry. CBOE continues to work with the FIX protocol organization and to participate in FIX working groups to help evolve the FIX protocol for wider use in exchange based derivatives markets.

Introduction to the CAS Simulator

The CAS Simulator is a stand-alone application that is written in Java. It simulates the operation of a CAS. Responses to a subset of operations have been provided to enable testing of applications written using the CMi.

Future versions of the CAS will be used for Stand Alone Testing as part of the application certification process.

Not all interfaces are supported by this version of the CAS Simulator.

Getting Started

The first step is to acquire the CAS Software Development Kit from the CBOE Website <http://systems.cboe.com/>. The next step is to install that software.

Changes for the CAS Simulator Version 2.0 are highlighted throughout the document.

Installing the CAS Simulator

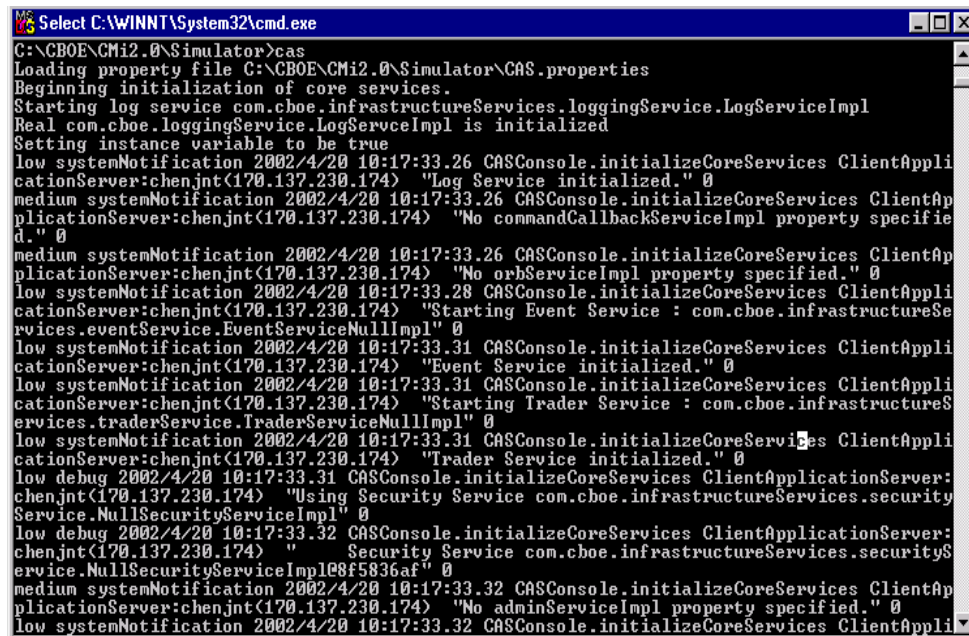
The CAS Simulator requires that Sun's JDK 1.4.2 from Sun Microsystems (<http://www.sun.com/products-n-solutions/software/oe-platforms/java2.html>) be installed on the computer where you are running the CAS.

The installation routine provided with the CAS Software Development Kit automatically installs the CAS software underneath the Install directory (default is C:\CBOE\CMi) in a directory named Simulator.

CAS Simulator Operation

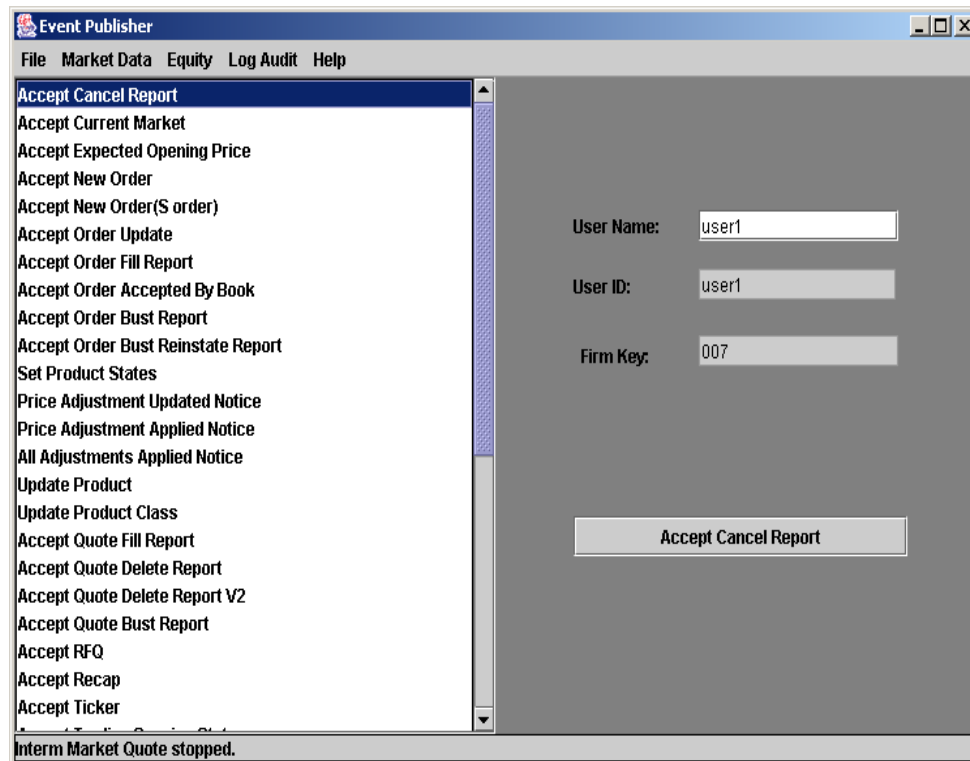
The simulator is started by running the CAS command file (CAS.BAT on Windows NT or CAS.sh on Solaris) that is installed in the *Simulator* directory. You can invoke this command by opening the file from a file explorer (Windows Explorer on Microsoft NT) or from a command line. If you are running from a command line—you should first make the *Simulator* directory your working directory.

Two windows will appear. The first window is a console log that shows messages produced by the CAS Simulator.



```
C:\CBOE\CMi2.0\Simulator>cas
Loading property file C:\CBOE\CMi2.0\Simulator\CAS.properties
Beginning initialization of core services.
Starting log service com.cboe.infrastructureServices.loggingService.LogServiceImpl
Real com.cboe.loggingService.LogServiceImpl is initialized
Setting instance variable to be true
low systemNotification 2002/4/20 10:17:33.26 CASConsole.initializeCoreServices ClientAppli
cationServer:chenjnt(170.137.230.174) "Log Service initialized." 0
medium systemNotification 2002/4/20 10:17:33.26 CASConsole.initializeCoreServices ClientAp
plicationServer:chenjnt(170.137.230.174) "No commandCallbackServiceImpl property specifie
d." 0
medium systemNotification 2002/4/20 10:17:33.26 CASConsole.initializeCoreServices ClientAp
plicationServer:chenjnt(170.137.230.174) "No orbServiceImpl property specified." 0
low systemNotification 2002/4/20 10:17:33.28 CASConsole.initializeCoreServices ClientAppli
cationServer:chenjnt(170.137.230.174) "Starting Event Service : com.cboe.infrastructureSe
rvices.eventService.EventServiceNullImpl" 0
low systemNotification 2002/4/20 10:17:33.31 CASConsole.initializeCoreServices ClientAppli
cationServer:chenjnt(170.137.230.174) "Event Service initialized." 0
low systemNotification 2002/4/20 10:17:33.31 CASConsole.initializeCoreServices ClientAppli
cationServer:chenjnt(170.137.230.174) "Starting Trader Service : com.cboe.infrastructureS
ervices.traderService.TraderServiceNullImpl" 0
low systemNotification 2002/4/20 10:17:33.31 CASConsole.initializeCoreServices ClientAppli
cationServer:chenjnt(170.137.230.174) "Trader Service initialized." 0
low debug 2002/4/20 10:17:33.31 CASConsole.initializeCoreServices ClientApplicationServer:
chenjnt(170.137.230.174) "Using Security Service com.cboe.infrastructureServices.securityS
ervice.NullSecurityServiceImpl" 0
low debug 2002/4/20 10:17:33.32 CASConsole.initializeCoreServices ClientApplicationServer:
chenjnt(170.137.230.174) "Security Service com.cboe.infrastructureServices.securityS
ervice.NullSecurityServiceImpl08f5836af" 0
medium systemNotification 2002/4/20 10:17:33.32 CASConsole.initializeCoreServices ClientAp
plicationServer:chenjnt(170.137.230.174) "No adminServiceImpl property specified." 0
low systemNotification 2002/4/20 10:17:33.32 CASConsole.initializeCoreServices ClientAppli
```

The second window is the CAS Event Publisher Window that is used to send events to client applications that are connected to the CAS Simulator and to shutdown the CAS Simulator.



Using the CAS Simulator

Once the CAS Event Publisher Window appears, you can start running client applications against the CAS. Once your application is running, you can then send events to it from the Event Publisher Window. Select the event you wish to run from the list on the left of the window, then click the button on the right to send the event to the simulator. In order to receive simulated events in the client application, you will have to match the user name field in the Event Publisher to the one the client application uses at logon. For simulated auction events, however, publishing an auction event will simulate an auction, but any user who is subscribed for auction events in the simulator will receive the events. The userID field provided for the auction event will be that which is in the user name field on the Event Publisher GUI. Currently, the events are always sent only for one product, i.e. IBM OCT 2007 100 CALL.

Starting Market Data

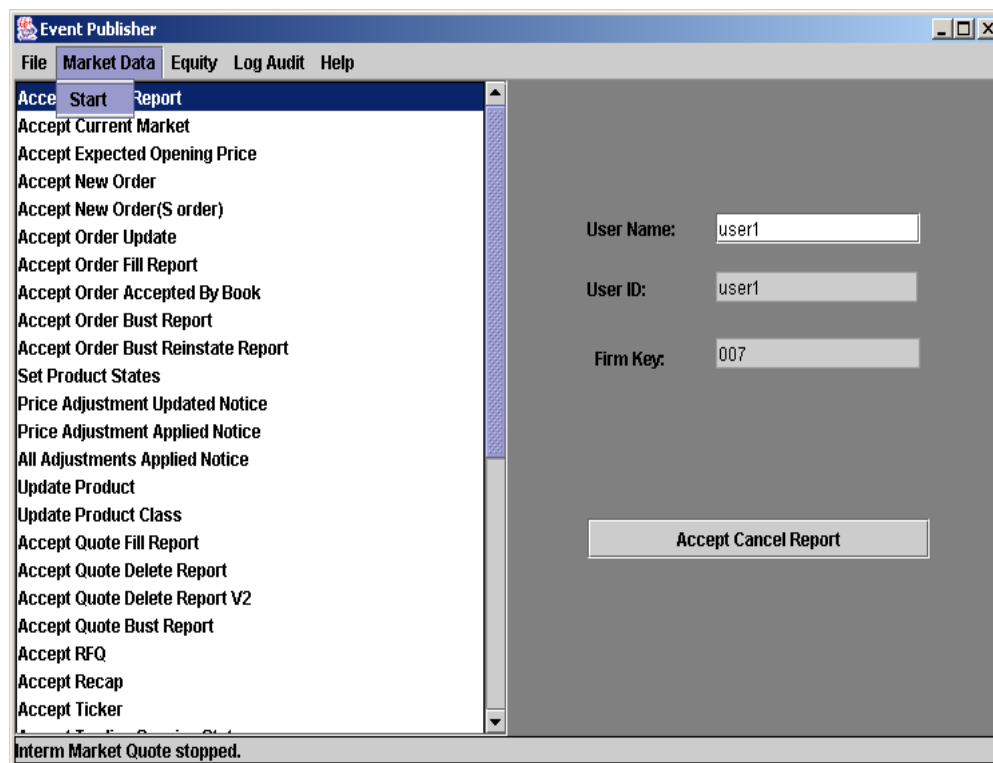
The CAS Event Publisher does not automatically generate market data events. To start (and stop) the publishing of market data—select the Market Data menu item shown in examples below.

The CAS simulator generates two different kinds of market data depending on the version of the MarketQuery interface. The existing Market Data option will generate data for versions 1, 2, 3 of MarketQuery. It will generate one product per session class for all classes. The second option, Start MDX Data, will generate market data for MarketQueryV4, CBOE's Market Data Express (MDX) data feed. All V4 callbacks will be called: Current Market, Recap, Last Sale

and Ticker. It will rotate calls through the series that are subscribed for by the client application. The Reset MDX Sequence menu item will reset the sequence number to 1 to help test client code for out of sequence messages.

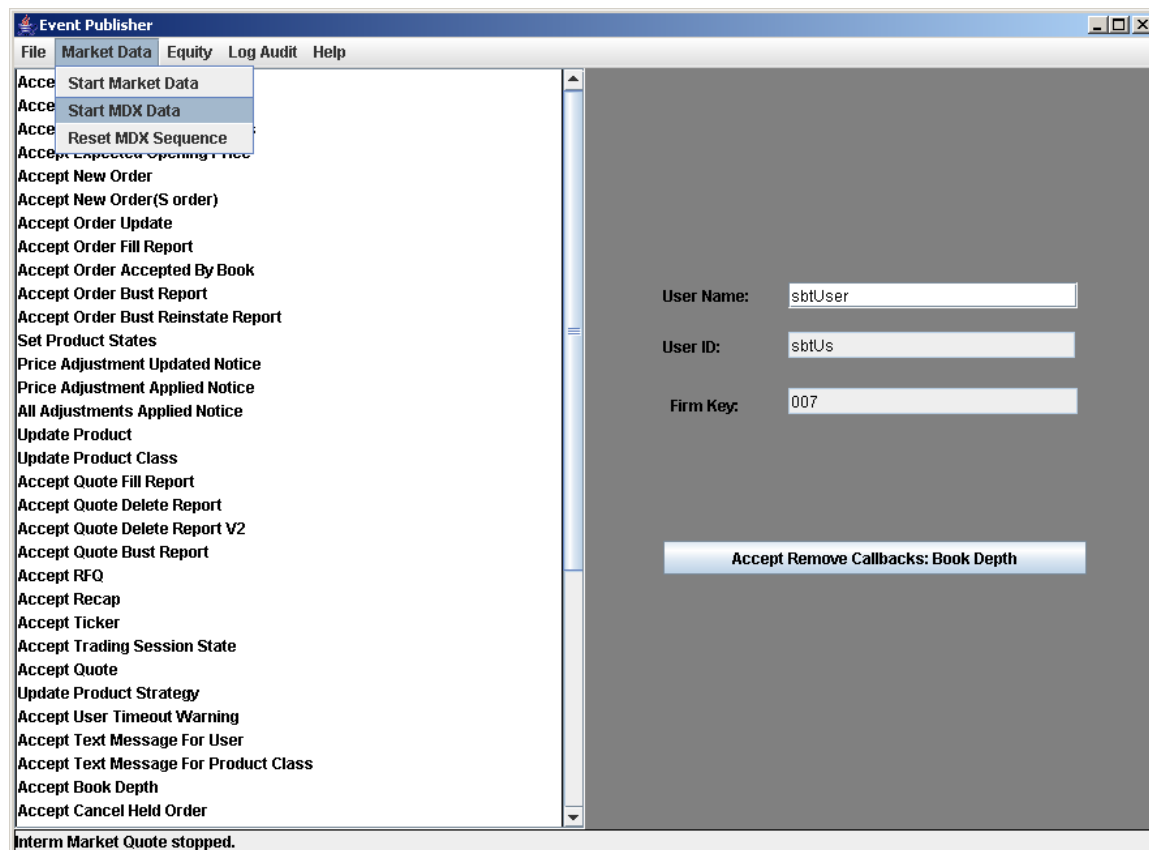
Below are two examples: (1) Market Data and (2) Market Data Express (MDX)

Example 1 – Generate Market Data



Market Data is getting generated for one product per session class for all classes. For example, for IBM Option Class, the recap/ticker will be started for product IBM JUL 2007 65.00 CALL. For IBM Future Class, the recap/ticker will be started for product IBM1C Dec 2002.

Example 21 – Generate MDX Data



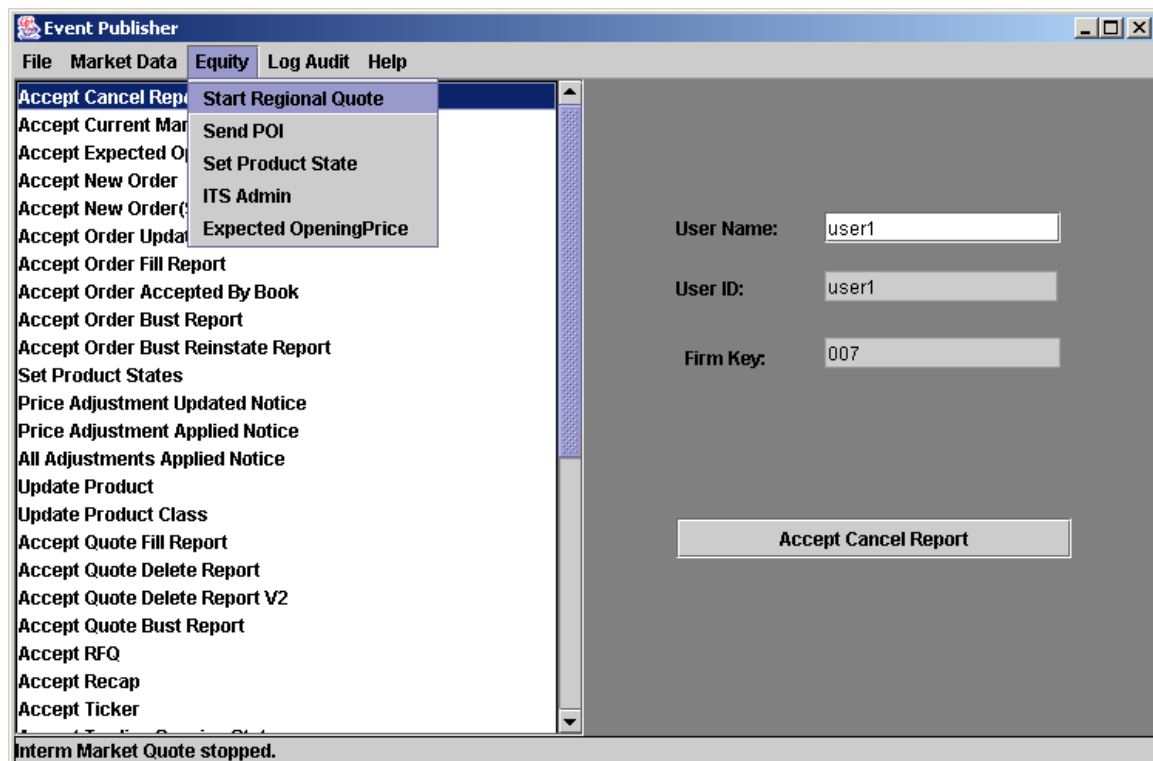
In the simulator, users can modify the number of messages sent in each block as well as the delay in milliseconds between publishes by modifying the following two lines in the cas.properties file in the simulator directory:

```
ClientApplicationServer.DefaultContainer.MDXGeneratorHome.generator_timeout=1000
ClientApplicationServer.DefaultContainer.MDXGeneratorHome.blockSize=10
```

Equity Menu

This menu has been added to support stock functionality in the simulator, which includes how to:

- Generate regional quotes and trades
- Send Pre opening price indication
- Set Product State
- Send ITS admin message
- Send expected opening price



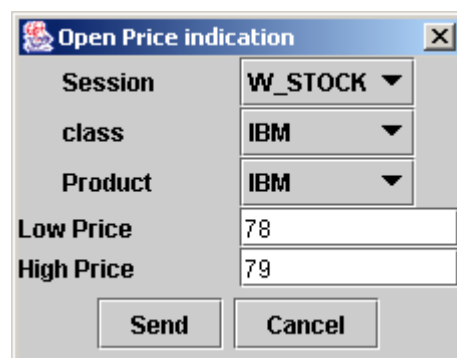
Starting Regional Quotes and Trades

This option is used to generate quotes and trades for all regional and primary stock exchanges.

- Go to the **Equity** menu, select **Start Regional Quote**

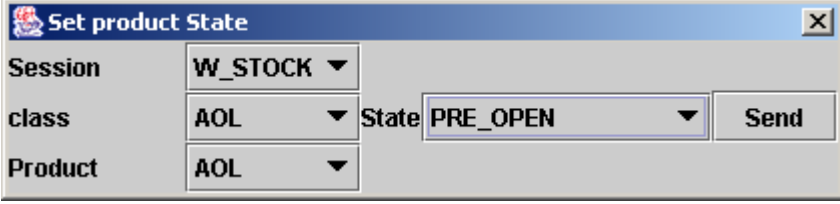
Sending Pre-Opening Price Indication

- Go to the **Equity** menu, select **Send POI**
- Select session, product class and product.
- Enter low price and high price



Sending Product State Updates

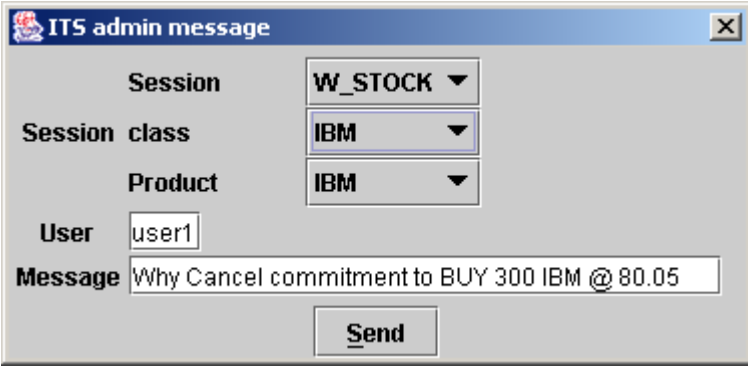
- Go to the **Equity** menu, select **Send Product State**
- Select session, product class and product
- Select product state



The 'Set product State' dialog box contains three dropdown menus: 'Session' with 'W_STOCK' selected, 'class' with 'AOL' selected, and 'Product' with 'AOL' selected. To the right of these is a 'State' dropdown menu with 'PRE_OPEN' selected. A 'Send' button is located to the right of the 'State' dropdown.

Sending ITS Admin Message


- Go to **Equity** menu, select **ITS Admin**
- Select session, product class and product
- Enter text message



The 'ITS admin message' dialog box includes dropdown menus for 'Session' (W_STOCK), 'Session class' (IBM), and 'Product' (IBM). It also has a 'User' text field containing 'user1' and a 'Message' text field containing 'Why Cancel commitment to BUY 300 IBM @ 80.05'. A 'Send' button is at the bottom right.

Sending Expected Opening Price

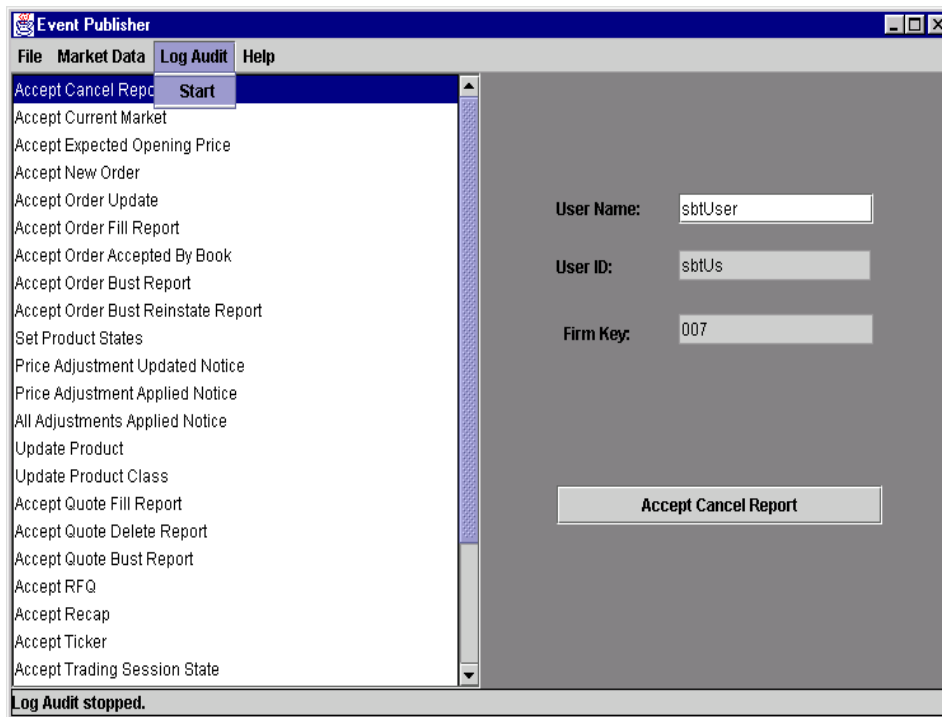
- Go to the **Equity** menu, select **Expected Opening Price**
- Select session, product class and product
- Enter expected opening price



The 'Expected Opening Price' dialog box features dropdown menus for 'Session' (W_STOCK), 'class' (IBM), and 'Product' (IBM). Below these is an 'EOP' text field with the value '78.50'. A 'Send' button is positioned at the bottom of the dialog.

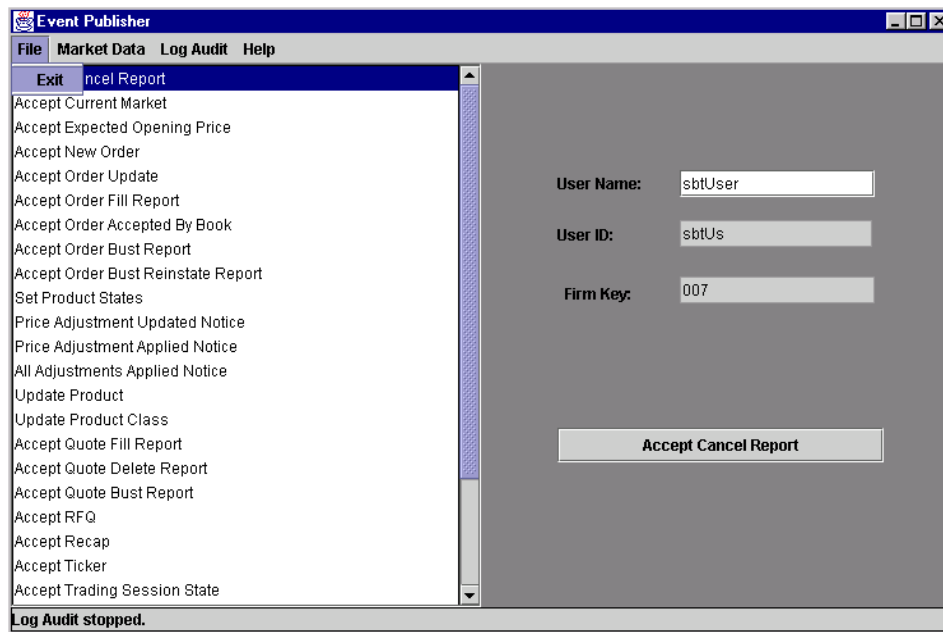
Starting Log Audit

The CAS Simulator does not automatically generate market data events. To start (and stop) the Audit Log—select the Log Audit menu item shown in the following.

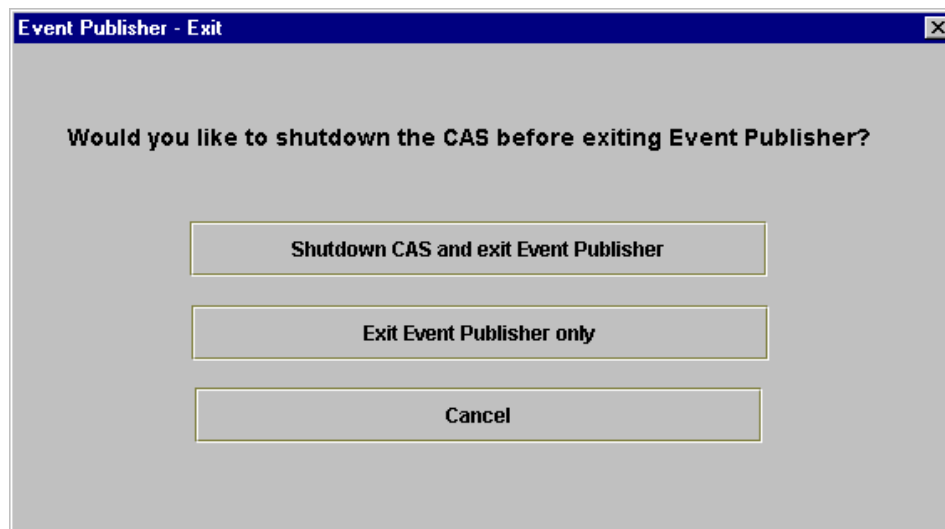


Stopping the CAS Simulator

Invoking the Exit command from the File menu in the CAS Event Publisher Window will present a window in which are prompted to stop the CAS Simulator, the CAS Event Publisher, or both.

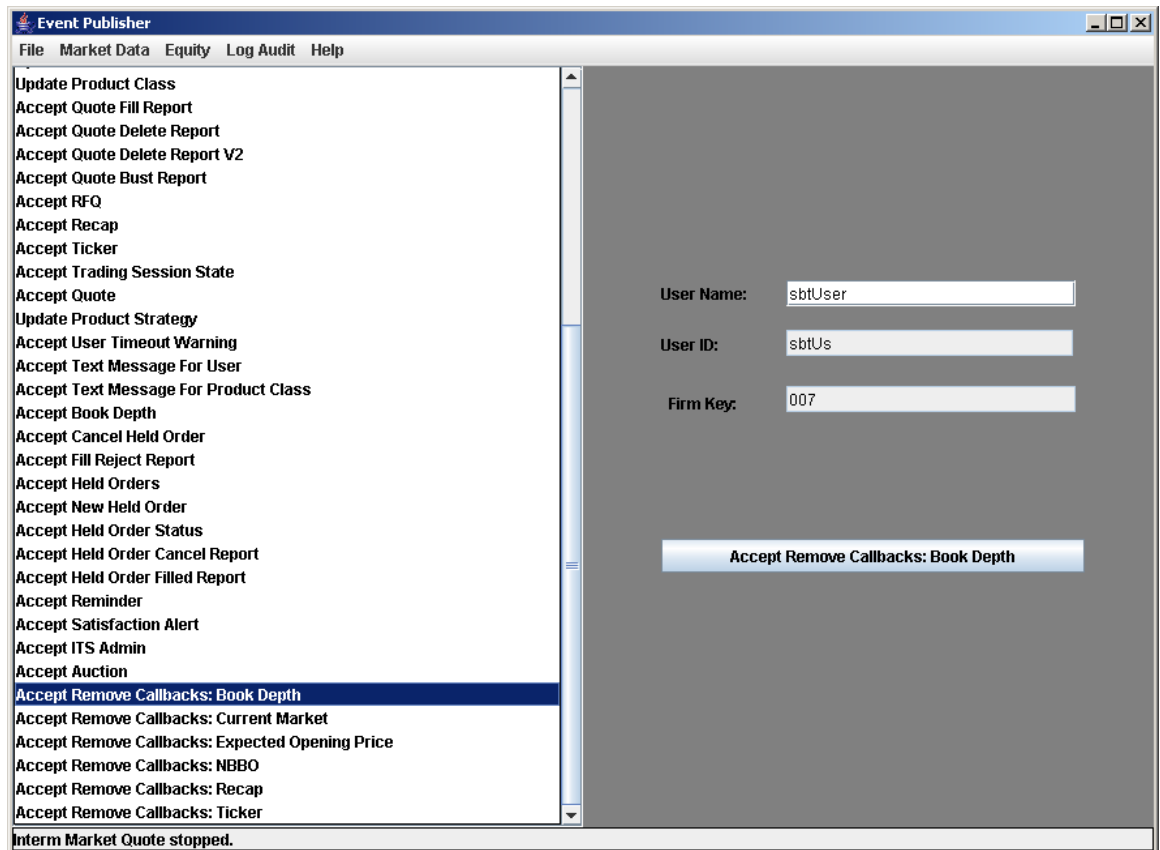


Under normal circumstances you would simply stop both the CAS and the Event Publisher. The option has been provided to stop the CAS Event Publisher only. This is useful, when you are testing a program and do not want your callbacks invoked by the CAS Simulator.



Callback Removal

The CAS Simulator allows users to remove callbacks from their running test programs.



Accept Remove Callbacks: Book Depth

Accept Remove Callbacks: Current Market

Accept Remove Callbacks: Expected Opening Price

Accept Remove Callbacks: NBBO

Accept Remove Callbacks: Recap

Accept Remove Callbacks: Ticker

Selecting one of these commands removes all callbacks of the corresponding type, as if all the relevant callback objects had failed during some callback. This includes notification via `CMIUserSessionAdmin.acceptCallbackRemoval()`. For example, let's assume your test program has 3 callback objects for Book Depth and 3 different callback objects for Current Market. If you click on Accept Remove Callbacks: Book Depth, the CAS Simulator will call `acceptCallbackRemoval()` and remove callbacks for all 3 Book Depth callback objects.

CAS Simulator Responses by Interface

The CAS Simulator has been developed to provide a predefined set of information to your application using the CMI. The information produced by the CAS Simulator is described by interface in the following sections.

Market Query Interface

Market Data Subscription

Market data published by the simulator is generated by the ticker simulation data feed – generating ticker and recap information for all products for each trading session. Market data is also generated as a result of simulated trading your client applications may be doing using the *Quote* or *Order Entry* interfaces of the simulator. The following subscription operations are used to subscribe for market data and are supported by the simulator. The *unsubscribe* operations will unsubscribe your application from market data.

subscribeRecap

subscribeRecapForClass

subscribeRecapForClassV2

subscribeRecapForProduct

subscribeRecapForProductV2

subscribeCurrentMarket

subscribeCurrentMarketForClass

subscribeCurrentMarketForClassV2

subscribeCurrentMarketForClassV3

subscribeCurrentMarketForProduct

subscribeCurrentMarketForProductV2

subscribeCurrentMarketForProductV3

subscribeNBBOForProduct

subscribeNBBOForProductV2

subscribeNBBOForClass

subscribeNBBOForClassV2

subscribeTicker

subscribeBookDepth

subscribeBookDepthV2

If top(n) of the book changes, a book depth struct that represents top(n) of the book will be published for this product for a given trading session. In simulator, n=5.

unsubscribeCurrentMarket

unsubscribeCurrentMarketForClass

unsubscribeCurrentMarketForClassV2

unsubscribeCurrentMarketForClassV3

unsubscribeCurrentMarketForProduct

unsubscribeCurrentMarketForProductV2

unsubscribeCurrentMarketForProductV3
unsubscribeNBBOForProduct
unsubscribeNBBOForProductV2
unsubscribeNBBOForClass
unsubscribeNBBOForClassV2
unsubscribeRecap
unsubscribeRecapForProduct
unsubscribeRecapForProductV2
unsubscribeRecapForClass
unsubscribeRecapForClassV2
unsubscribeBookDepth
unsubscribeBookDepthV2
unsubscribeTicker

The results are generated for all defined products in the following manner. A process thread is created for all trading sessions and classes supported by the simulator. This thread generates a market data stream for the first product of the individual class. For IBM options it will generate market data for the first series listed. The base starting values for last sale-generated data are:

Date and times are from machine current date and time.

The product data structures are set from the published product information in the Product Query Interface

The last sale values are based off the initial tick information of:

- last sale price = 5.0(options) 95.0(stocks);
- increment = 0.25;
- low = 4.0(options) 85.0(stocks);
- high = 6.25(options) 105.0(stocks);
- last Sale Volume = 10;
- total Volume = 10;
- tick Direction = '+';
- net Change = 0.0;
- bid Price = 4.9;
- bid Size = 1;
- ask Price = 5.1;
- ask Size = 1;
- recap Prefix = "C";
- tick = 5.0;
- low Price = 1.0;
- high Price = 11.25;
- open Price = 5.0;
- close Price = 11.25;
- open Interest = 1;
- previous Close Price = 5.0;

As each 'tick' goes by the last sale price moves by +0.25 increments up to 6.25 then down by -0.25 until 4.0. The information published is adjusted as follows:

TICKER CHANGE

```
ticker.base.lastSalePrice = doubleToPriceStruct( tic.getValue() );  
ticker.base.lastSaleVolume += 10;
```

RECAP CHANGE

```
recap.base.lastSalePrice      = doubleToPriceStruct( tic.getValue() );  
recap.base.lastSaleVolume    = ticker.base.lastSaleVolume;  
recap.base.totalVolume       = ticker.base.lastSaleVolume;  
recap.base.tickDirection     = '+';  
if ( tic.getIncrement() < 0 )  
{  
    recap.base.tickDirection = '-';  
}  
recap.base.netChange = doubleToPriceStruct( tic.getValue()- tic.getStart());  
recap.base.bidPrice  = doubleToPriceStruct( tic.getValue() - (double)0.1 );  
recap.base.bidSize   = 1;  
recap.base.askPrice  = doubleToPriceStruct( tic.getValue() + (double)0.1 );  
recap.base.askSize   = 1;  
recap.base.recapPrefix = recap.productInformation.productSymbol;  
recap.base.tick       = doubleToPriceStruct( tic.getValue() );  
recap.base.lowPrice   = doubleToPriceStruct( tic.getValue()- tic.getMin());  
recap.base.highPrice  = doubleToPriceStruct( tic.getValue()+ tic.getMax());  
recap.base.openPrice  = doubleToPriceStruct( tic.getStart());  
recap.base.closePrice = doubleToPriceStruct( tic.getValue()+ tic.getMax());  
recap.base.openInterest = 1;  
recap.base.previousClosePrice = doubleToPriceStruct( tic.getValue() );
```

Out of Sequence Trade

There are only two conditions under which a trade would be reported out of sequence: (1) Block Trades and (2) EFPs (Exchange for Physicals). The Ticker and Recap prefix fields will identify Block trades as "BLKT" and EFPs as "EXPH".

subscribeExpectedOpeningPrice

Not implemented at this time.

unsubscribeExpectedOpeningPrice

Not implemented at this time.

getBookDepth

Returns a book depth struct that represents a summary of all orders currently in the book for this product for a given trading session.

getDetailMarketDataHistoryByTime

Not implemented at this time.

getPriorityMarketDataHistoryByTime

Not implemented at this time.

getMarketDataHistoryByTime

- The market data returned is a fixed 10 record history. The "tradePrice" for each record increases by one from 50 to 59. The "bidPrice" from 49 to 58. The "askPrice" from 51 to 60.
- The first history record fields values are:

```
tradePrice = PriceStruct(PriceTypes.MARKET, 10, 50);
tradeQuantity = 10;
sellerAcronym = "ABC";
buyerAcronym = "XYZ";
bidSize = 10;
bidPrice = new PriceStruct(PriceTypes.MARKET, 10, 49);
askSize = 15;
askPrice = PriceStruct(PriceTypes.MARKET, 10, 51);
underlyingLastSalePrice = PriceStruct(PriceTypes.MARKET,100,50);
```

- The second history record fields values are:

```
tradePrice = PriceStruct(PriceTypes.MARKET, 10, 51);
tradeQuantity = 10;
sellerAcronym = "ABC";
buyerAcronym = "XYZ";
bidSize = 10;
bidPrice = PriceStruct(PriceTypes.MARKET, 10, 50);
askSize = 15;
askPrice = PriceStruct(PriceTypes.MARKET, 10, 52);
underlyingLastSalePrice = PriceStruct(PriceTypes.MARKET,100,51);
```

- The last history record fields values are:

```
tradePrice = PriceStruct(PriceTypes.MARKET, 10, 59);
tradeQuantity = 10;
sellerAcronym = "ABC";
buyerAcronym = "XYZ";
bidSize = 10;
bidPrice = PriceStruct(PriceTypes.MARKET, 10, 58);
askSize = 15;
askPrice = PriceStruct(PriceTypes.MARKET, 10, 60);
underlyingLastSalePrice = PriceStruct(PriceTypes.MARKET,100,59);
```

Order Entry Interface

The following fields are the important differences between the different types of orders. All remaining fields are filled in as per the individual order's trading requirements.

- Limit Order:
price type = com.cboe.idl.cmiConstants.PriceTypes.LIMIT
contingency type = com.cboe.idl.cmiConstants.ContingencyTypes.NONE
contingency order price type = com.cboe.idl.cmiConstants.PriceTypes.LIMIT
contingency order volume = 0
- Market Order:
price type = com.cboe.idl.cmiConstants.PriceTypes.MARKET
contingency type = com.cboe.idl.cmiConstants.ContingencyTypes.NONE
contingency order price type = com.cboe.idl.cmiConstants.PriceTypes.NO_PRICE
contingency order volume = 0
- Stop Order:
price type = com.cboe.idl.cmiConstants.PriceTypes.MARKET
contingency type = com.cboe.idl.cmiConstants.ContingencyTypes.STP
contingency order price type = com.cboe.idl.cmiConstants.PriceTypes.VALUED
contingency order volume = 0
- Stop Limit Order:
price type = com.cboe.idl.cmiConstants.PriceTypes.LIMIT
contingency type = com.cboe.idl.cmiConstants.ContingencyTypes.STP_LOSS

```
contingency order price type =  
com.cboe.idl.cmiConstants.PriceTypes.VALUED  
contingency order volume = 0
```

Strategy Products

- Stop and Stop_Loss Orders
 - Order Price: MARKET
 - Contingency Field: Stop Price
- Stop Limit Order
 - Order Price: Limit Price
 - Contingency Field: Stop Price

When strategy products, i.e. spreads, are in the ON_HOLD state, any order for that strategy product will either trade directly with the orders on the strategy legs or will be put in the order book. In addition, strategy products that are in the ON_HOLD state do not trade strategy orders against other strategy orders, but can still trade strategy orders against orders for the individual legs products.

acceptOrder

- Support for Auction Response order. Contingency type must be ContingencyTypes.AUCTION_RESPONSE. Extensions field must contain AUCTION_ID=<cboe_high_id>:<cboe_low_id>
 - All standard acceptOrder validations apply.
 - If an order with the same order ID fields already exists, a DataValidationException will be thrown with error code DataValidationCodes.DUPLICATE_ID.
 - If the extensions field cannot be parsed, a DataValidationException will be thrown with error code DataValidationCodes.INVALID_EXTENSIONS.
 - If the extensions field is present and parseable and the auction id value cannot be parsed, or the auction id does not exist, a DataValidationException will be thrown with error code DataValidationCodes.INVALID_AUCTION_ID.
 - If the auction response order has a side value that is not BUY or SELL, or if the side value is not the opposite side of the primary (customer) order in the auction, a DataValidationException will be thrown with error code DataValidationCodes.INVALID_SIDE.
 - If the session for the auction response order is not equal to W_MAIN, a DataValidationException will be thrown with error code DataValidationCodes.INVALID_CONTINGENCY_TYPE.
 - If the product key for the auction response order does not match the product key of the primary order, a DataValidationException will be thrown with error Code DataValidationCodes.INVALID_PRODUCT.
 - If the auction has expired, a NotAcceptedException will be thrown with NotAcceptedCodes.AUCTION_ENDED.

- Support for Linkage Satisfaction (S) order.
 - If the destinationExchange is empty or CBOE, a dataValidationException with error code of DataValidationCodes.INVALID_EXCHANGE is thrown.
 - If the order contingency type is not IOC, a DataValidationException with error code of DataValidationCodes.INVALID_CONTINGENCY_TYPE is thrown.
 - If the S order entering time is longer than 3 minutes after the alert is sent, a dataValidationException with error code of DataValidationCodes.INVALID_TIME is thrown.
 - If the S order quantity is greater than trade through quantity, a dataValidationException with error code of DataValidationCodes.INVALID_QUANTITY is thrown.
 - If the destination exchange is not the exchange traded through, a dataValidationException with error code of DataValidationCodes.INVALID_EXCHANGE is thrown.
 - S orders will not be booked, however, a new order status will still be sent back to NBBO agent.
 - Order status extensions field will contain tradeThroughPrice, tradeThroughQuantity and tradeThroughTime in addition to awayExchange and alertId that is entered by the user.
- Support for Linkage Principal (P) order.
- Orders for IBM and DJX will be half filled and half booked.
- Orders for AOL, OEX and all Future products will be fully filled.
- Orders for GM will be booked.
- The CAS.properties to be able to configure products in a product class to be traded half filled/half booked, all filled, or all booked.
- A filled report will be sent to subscribers for OrderStatus.
- Recap and Ticker data for the newly filled order will be for subscribers for recap and ticker for the product that was ordered.
- If the order quantity is less than 1 a DataValidationException (errorCode = DataValidationCodes.INVALID_QUANTITY) is thrown.
- If the order sessionNames length equals 0 a DataValidationException (errorCode = DataValidationCodes.INVALID_SESSION) is thrown.
- If the valued order price is not valid a DataValidationException (errorCode = DataValidationCodes.INVALID_PRICE) is thrown.
- If the product key in the submitted order does not exist a notAcceptedException (errorCode = DataValidationCodes.INVALID_PRODUCT) is thrown.

- If the orderId structure is a duplicate a `DataValidationException` (`errorCode = DataValidationCodes.DUPLICATE_ID`) is thrown. Restarting the CAS removes all orders.

acceptInternalizationOrder

- A primary (customer) order and a match (firm) order for a non-strategy product can be submitted to start an auction by calling this method and specifying match type of `MatchTypes.LIMIT` or `MatchTypes.AUTO_MATCH`.
- If any of the normal validations for the primary (customer) order fail validation, the exception will be rethrown and the primary order will not be auctioned.
- If the standard validations for the primary order succeed, the following additional validations will be performed:
 - If the session name is not `W_MAIN`, a `DataValidationException` will be thrown with error code `DataValidationCodes.INTERNALIZATION_NOT_ALLOWED`.
 - If the product in the order is not an option product, a `DataValidationException` will be thrown with error code `DataValidationCodes.INVALID_PRODUCT_TYPE`.
 - If the primary order has a contingency of `ContingencyTypes.AUCTION_RESPONSE`, a `DataValidationException` will be thrown with error code `DataValidationCodes.INVALID_CONTINGENCY_TYPE`.
 - If the primary order has price type other than `PriceTypes.LIMIT` or `PriceTypes.MARKET`, a `DataValidationException` will be thrown with error code `DataValidationCodes.INVALID_PRICE`.
 - If the above validations for the primary order succeed, the following validations will be performed on the match type and match order. Any validation failure will result in the primary order being traded as a normal order. In these cases, the `InternalizationOrderResultStruct` will contain an `OrderResultStruct` with a valid order ID for the primary order and an error code and error message for the match order.
 - If the standard order validation fails on the match order, the error code and message corresponding to the validation failure will be returned in the `OrderResultStruct` for the match order.
 - If the primary and match order have the same order ID fields, or an order with the same order id fields has already been accepted, the match order will be rejected with `DataValidationCodes.DUPLICATE_ID`.
 - If the match type is `MatchTypes.LIMIT` and the price of the match order is not a limit price, then the match order will be rejected with `DataValidationCodes.INVALID_PRICE`.
 - If the match type is `MatchTypes.AUTO_MATCH` and the price of the match order is not a market price, then the match order will be rejected with `DataValidationCodes.INVALID_PRICE`.
 - If the match type neither `MatchTypes.LIMIT` or `MatchTypes.AUTO_MATCH`, then the match order will be rejected with `DataValidationCodes.INVALID_MATCH_TYPE`.

- If the match order contingency type is `ContingencyTypes.AUCTION_RESPONSE`, then the match order will be rejected with `DataValidationCodes.INVALID_CONTINGENCY`.
- If the match order side is not `Sides.BUY` or `Sides.SELL`, or the match order side is not the opposite side of the primary order, then the match order will be rejected with `DataValidationCodes.INVALID_SIDE`.
- If the price of the match order is not at or an improvement of the price for the primary order, then the match order will be rejected with `DataValidationCodes.INVALID_PRICE`.
- If the match order quantity is less than the quantity of the primary order, then the match order will be rejected with `DataValidationCodes.INVALID_QUANTITY`.

acceptInternalizationStrategyOrder

- A primary (customer) order and a match (firm) order for a strategy product can be submitted to start an auction by calling this method and specifying match type of `MatchTypes.LIMIT` or `MatchTypes.AUTO_MATCH`.
- If any of the normal validations for the primary (customer) order fail validation, the exception will be rethrown and the primary order will not be auctioned.

acceptOrderCancelRequest

- The `acceptCancelReport` event will be sent out for the cancelled order through the event channel.
- When the product key in the submitted order does not exist a `DataValidationException` (`errorCode = DataValidationCodes.INVALID_PRODUCT`) is thrown.
- When the order is not found a `DataValidationException` (`errorCode = DataValidationCodes.INVALID_ORDER_ID`) is thrown.
- When `userId` is different from the original order submitter `userId` a `DataValidationException` (`errorCode = AuthenticationCodes.UNKNOWN_USER`) is thrown.
- When the quantity specified is greater than the remaining quantity or quantity equals 0 a `DataValidationException` (`errorCode = DataValidationCodes.INVALID_QUANTITY`) is thrown.
- Support for Auctions
 - The standard cancel request validations apply to all auction participant orders. Only auction response orders can be canceled.
 - If the cancel request is for a primary (customer) order, the request will be accepted and a Too Late to Cancel will be published with cancel reason `ActivityReasons.USER`.

- If the cancel request is for a match order, a `NotAcceptedException` will be thrown with `NotAcceptedCodes.INVALID_REQUEST`.
- If the cancel request is for an auction response order, and the auction has expired, a `NotAcceptedException` will be thrown with `NotAcceptedCodes.AUCTION_ENDED`.

acceptOrderUpdateRequest

- Only update the following 3 fields: `theOrder.optionalData`, `theOrder.account`, `theOrder.cmta`, other field updates will be ignored.
- The `acceptOrderUpdate` event will be sent out for the updated order through the event channel.
- When the order is not found a `DataValidationException` (`errorCode = DataValidationCodes.INVALID_ORDER_ID`) is thrown.
- When the quantity specified does not match the remaining quantity or quantity equals 0 a `DataValidationException` (`errorCode = DataValidationCodes.INVALID_QUANTITY`) is thrown.

acceptOrderCancelReplaceRequest

- The `acceptCancelReport` event will be sent out for the replaced order through the event channel.
- If the order `sessionNames` length equals 0 a `DataValidationException` (`errorCode = DataValidationCodes.INVALID_SESSION`) is thrown.
- When the new order uses the same order ID as the to-be-replaced order ID a `DataValidationException` (`errorCode = DataValidationCodes.DUPLICATE_ID`) is thrown.
- When the order to be replaced is not found a `DataValidationException` (`errorCode = DataValidationCodes.INVALID_ORDER_ID`) is thrown.
- When the quantity specified does not match with the remaining quantity a `DataValidationException` (`errorCode = DataValidationCodes.INVALID_QUANTITY`) is thrown.
- Support for Auctions
 - The same validations for cancel requests for auction orders will apply to cancel replace requests. The replacement order will be subject to the standard `acceptOrder` validations for auction orders. The cancel request will not be processed if the replacement order fails validation.

acceptCrossingOrder

- The fill reports for both orders will be published. The following validation failure will result in a `DataValidationException` to be thrown:
 1. one or both order already exist. Error Code = `DataValidationCodes.DUPLICATE_ID`.
 2. crossing orders not on the same product. Error Code = `DataValidationCodes.INVALID_PRODUCT`.
 3. invalid product used. Error Code = `DataValidationCodes.INVALID_PRODUCT`.

4. crossing orders not on different side. Error Code = `DataValidationCodes.INVALID_SIDE`.
5. crossing orders not on the same price. Error Code = `DataValidationCodes.INVALID_PRICE`.
6. Crossing order quantity mismatch. Error Code = `DataValidationCodes.INVALID_QUANTITY`.

acceptRequestForQuote

- No simulated data transactions.
- Any entered RFQ data is accepted and disseminated to all subscribers.
- If the product does not exist a `DataValidationException` (errorCode = `DataValidationCodes.INVALID_PRODUCT`) is thrown.

acceptOrderByProductName

- Create and submit a Linkage P order.
- Query the product based on product name. If the product does not exist a `DataValidationException` (errorCode = `DataValidationCodes.INVALID_PRODUCT`) is thrown.
- It includes all the events `acceptOrder` has.

acceptStrategyOrder

- Handles strategy Order.
- In simulator, `acceptStrategyOrder` works the same as `acceptOrder`.

acceptStrategyOrderUpdateRequest

- The `acceptOrderUpdate` event will be sent out for the updated strategy order through the event channel.
- In simulator, it works the same as `acceptOrderUpdateRequest`

acceptStrategyOrderCancelReplaceRequest

- In simulator, it works the same as `acceptStrategyOrderCancelReplaceRequest`.

Order Query Interface

getPendingAdjustmentOrdersByProduct

- Returns the users current orders for the given product and trading session.

getPendingAdjustmentOrdersByClass

- Returns the users current orders for the given class and trading session.

queryOrderHistory

- Returns a single history struct showing the order has been accepted by the system as a booked order.

Order Query Get Operations

The simulator supports the following *get* operations. The CAS Simulator does not save order data in between testing sessions. The only order data available are the orders entered by the tester during the current testing session.

getOrdersForProduct

getOrdersForSession

getOrderById

getOrdersForType

getOrdersForClass

Order Data Subscription

All subscribe request methods register the supplied status consumer object for the order status information requested. Unsubscribe methods remove the supplied consumer from the publication list. The simulator will deliver any relevant data generated by other user order activity to the subscribed objects.

subscribeOrders

subscribeOrdersByFirm

subscribeOrdersWithoutPublish

subscribeOrdersByFirmWithoutPublish

subscribeAuction

unsubscribeAuction

unsubscribeOrderStatusForProduct

unsubscribeOrderStatusForSession

unsubscribeOrderStatusForFirm

unsubscribeAllOrderStatusForType

unsubscribeOrderStatusByClass

Product Definition Interface

acceptStrategy

- Simulator supports the creation of strategy on all Option products and two Future product classes, AXP and AIG.

buildStrategyRequestByName

- Permits you to specify a strategy type (such as straddle or time), specify an anchor product by name and a price and month increment for the subsequent legs of the strategy
- Simulator supports following strategy types. If a strategy type is not supported, a `DataValidationException(errorCode =0)` is thrown.
 - COMBO
 - DIAGONAL

- PSEUDO_STRADDLE
 - RATIO
 - STRADDLE
 - TIME
 - UNKNOWN
 - VERTICAL
 - BUY_WRITE
- Simulator always returns a strategy request with 2 legs. The first leg is the anchor product. The second leg is constructed based on the first leg and the strategy type.
 - If the product does not exist a `DataValidationException` is thrown.
- buildStrategyRequestByProductKey**
- Same as `buildStrategyRequestByName()` except uses a product key instead of a product name.
 - If the product does not exist a `DataValidationException` is thrown.

User History Interface

User History Get Operations

The simulator supports the following *get* operations for user history. Each history method will return a single Activity History Struct for a given trading session and product or class. Each struct will contain 19 activity history records, one for each activity history type defined in the CMI.

getTraderClassActivityByTime

getTraderProductActivityByTime

Product Query Interface

These methods return the requested product data as statically defined in the simulator product repository. The product query interface returns product information independent of the trading session. The actual data is outlined at the end of this section.

getProductTypes

getProductClasses

getProductsByClass

getProducts

getProductByName

getProductNameStruct

getProductClassByKey

getProductByKey
getClassByKey
getClassBySymbol
isValidProductName

Get Pending Adjustment Data

The following methods return all existing products as pending an adjustment.

getAllPendingAdjustments
getPendingAdjustments
getPendingAdjustmentProducts

Retrieve Strategy Information

The following methods are used for strategy retrieval

getStrategyByKey

- Returns all strategies that contains a given product key.
- If the strategy productKey is not found, NotFoundException will be thrown.

getStrategiesByClass

- Returns all strategies that contains a given class key

getStrategiesByComponent

- Return all strategies that contain a particular product as a leg

Products Defined in the CAS Simulator

There are five configured Option Classes for the simulator, IBM, AOL, GM, DJX and OEX. Also, the following Future Classes are added in this version of simulator:

AXP, AIG, AMGN, AMR, AMAT, T, BAC, ONE, BBY, BGEN, BMY, BRCM, BRCD, CEPH, CHKP, CVX, CSCO, C, KO, DELL, EBAY, EMC, EMLX, XOM, F, GE, GM, GENZ, GS, HAL, HD, IDPH, INTC, INVN, JPM, JNJ, KLAC, KKD, MRK, MER, MU, MSFT, MWD, MOT, NEM, NOK, NOC, NVLS, ORCL, PEP, PFE, MO, PG, QLGC, QCOM, SBC, SLB, SEBL, PCS, SBUX, SUNW, SYMC, TXN, TYC, UAL, VRTS, VZ, WMT, XLNX

The series information for each of these classes is stored in SimulatorData_ProductClasses.txt, SimulatorData_Products.txt, SimulatorData_ProductTypes.txt, SimulatorData_ReportingClasses.txt and SimulatorData_Strategies.txt. Simulator parses the data defined in these file and creates the products for simulator uses.

Trading Session Interface

The simulator emulates three configured trading sessions. Each trading session has been configured to support its own trading classes. All session related events generated by the Event Simulator dialog box will be generated for the simulator supported trading session.

getCurrentTradingSessions

- Returns all currently defined trading sessions. The current simulator returns four sessions with the trading session name of W_MAIN, W_U1, ONE_MAIN and W_STOCK.

W_MAIN is configured to trade OPTION and STRATEGY products. W_U1 is configured to trade INDEX, EQUITY or other underlying products. Currently, the OPTION classes traded in W_MAIN session are GM, AOL, IBM, DJX and OEX. The default trading session is W_MAIN. The ONE_MAIN session is configured to trade FUTURE and STRATEGY products. The W_STOCK session is configured to trade EQUITY only.

The following methods will return the data for the trading session from the defined product repository.

getProductTypesForSession

getClassesForSession

getProductsForSession

getStrategiesByClassForSession

getProductBySessionForKey

getStrategyBySessionForKey

getClassBySessionForKey

getProductBySessionForName

getClassBySessionForSymbol

getStrategiesByComponent

- Return all strategies in a trading session that contain a particular product as a leg

unsubscribeClassesByTypeForSession

unsubscribeProductsByClassForSession

unsubscribeStrategiesByClassForSession

unsubscribeTradingSessionStatus

Quote Interface

The quote processing simulator does not save quote data in between testing sessions. The only quote data available are the quotes entered by the tester during the current testing session. To enter a valid quote, the user has to send in both-sides quote. The quote with a bid side of zero quantity at zero price and a valid ask side is a valid quote.

acceptQuote

- If bid price is greater than or equal to ask price, a `DataValidationException(errorCode=DataValidationCodes.INVALID_PRICE)` is thrown.

- If a quote has a zero quantity and a non-zero price at the same side, a `DataValidationException(errorCode=DataValidationCodes.INVALID_QUANTITY)` is thrown.
- Half of the quote quantity will be booked.
- Half of the quote quantity will be filled.
- A filled report will be sent back through the event channel to all registered listeners.
- Market Data is updated and published to interested parties.
- If the product key in the submitted order does not exist a `DataValidationException (errorCode = DataValidationCodes.INVALID_PRODUCT)` is thrown.
- If the quantity is invalid a `DataValidationException (errorCode = DataValidationCodes.INVALID_QUANTITY)` is thrown.

acceptQuotesForClass

- The `acceptQuote` events are repeated as applicable for all supplied quotes.

acceptQuotesForClassV3

getQuote

- Retrieves current quote for given product..

cancelQuote

- Cancels current quote for given product.

cancelAllQuotes

- Cancels all current quotes for the user.

cancelAllQuotesV3

cancelQuotesByClass

- Cancels current quotes for the class.

Quote Subscription Operations

All subscribe request methods register the supplied status consumer object for the quote status information requested. Unsubscribe methods remove the supplied consumer from the publication list. The simulator will deliver any relevant data generated by other user quote activity to the subscribed objects.

subscribeQuoteStatus

unsubscribeQuoteStatus

subscribeRFQ

unsubscribeRFQ

subscribeQuoteStatusForFirm

unsubscribeQuoteStatusForFirm

subscribeQuoteStatusWithoutPublish

IntermarketQuery Interface

getIntermarketByProductForSession

- Returns a single `CurrentIntermarketStruct` for given product and trading session

getIntermarketByClassForSession

- Returns sequence of CurrentIntermarketStruct for give class and trading session.

Note: Simulator only returns hard coded CurrentIntermarketStruct

getAdminMessage

- Simulator will return ITS admin message kept in memory for the given session, product, message type and exchange. All message type will return if message type is zero. If exchange is space then message for all exchange will return.

getDetailedOrderBook

- Return all orders in the book

ShowMarketableOrderBookAtPrice

- Show the marketable orders in the order book at a given price

getOrderBookStatus

- Get the status of the order book if it is locked or not.

IntermarketHeldOrderEntry Interface

rerouteHeldOrder

- NBBOAgent uses this method to reroute held orders back to the system. In the simulator, the reroute will always succeed.

rerouteHeldOrderByClass

- NBBOAgent uses this method to reroute all held order for a given class. In the simulator, the reroute will always succeed.

acceptCancelResponse

- NBBOAgent uses this method to acknowledge the cancel request coming from the system. In the simulator, this acceptCancelResponse will always succeed. Following that, a heldOrderStatus event and an heldCancelCancelReport event will be published to this NBBOAgent.

acceptFillHeldOrder

- NBBOAgent uses this method to fill an order that is held by this NBBOAgent. In the simulator, this acceptFillHeldOrder will always succeed. Following that, heldOrderStatus and heldOrderFilledReport will be published to this NBBOAgent.

IntermarketManualHandling Interface

rerouteHeldOrder

- NBBOAgent uses this method to reroute held orders back to the system. In the simulator, the reroute will always succeed.

rerouteHeldOrderByClass

- NBBOAgent uses this method to reroute all held order for a given class. In the simulator, the reroute will always succeed.

acceptCancelResponse

- NBBOAgent uses this method to acknowledge the cancel request coming from the system. In the simulator, this acceptCancelResponse will always succeed. Following that, a heldOrderStatus event and an heldCancelCancelReport event will be published to this NBBOAgent.

acceptFillHeldOrder

- NBBOAgent uses this method to fill an order that is held by this NBBOAgent. In the simulator, this acceptFillHeldOrder will always succeed. Following that, heldOrderStatus and heldOrderFilledReport will be published to this NBBOAgent.

getHeldOrderById

- Returns an HeldOrderDetailStruct for given orderId.

lockProduct

- simulator always returns true when this method is called.

UnlockProduct

- simulator always returns true when this method is called.

rerouteBookedOrderToHeldOrder

- simulator always returns true when this method is called.

acceptSatisfactionOrderFill

- Not implemented in simulator

acceptSatisfactionOrderInCrowdFill

- Not implemented in simulator

acceptSatisfactionOrderReject

- Not implemented in simulator

acceptCustomerOrderSatisfy

- Not implemented in simulator

acceptFillReject

- Not implemented in simulator

getAssociatedOrders

- Not implemented in simulator

getOrdersByOrderTypeAndClass

- Not implemented in simulator

getOrdersByOrderTypeAndProduct

- Not implemented in simulator

acceptPreOpeningIndication

- Accept opening price for product, simulator accept the message and change the product state to OPEN

acceptPreOpeningResponse

- Simulator will convert the pre opening response into admin message kept it in memory.

acceptAdminMessage

- Simulator will keep the message in memory.

acceptOpeningPriceForProduct

- Accept opening price for product, simulator accept the message and change the product state to OPEN

NBBOAgentSessionManager Interface

getIntermarketManualHandling

- Returns IntermarketManualHandling interface.

NBBOAgent Interface

registerAgent

- NBBOAgent uses this method for NBBOAgent registration. An NBBOAgentSessionManager interface will be returned upon successful registration.
- Simulator checks if the session name and class key the NBBO agent is interested in is registered by any other NBBO agent. If not, the simulator will let the NBBO agent successfully register.
- If another NBBOAgent has already registered for the given session name and class key, the simulator will check the forcedtakeover flag in the current NBBOAgent. If the forcedtakeover flag is true, the simulator will let the current NBBOAgent force take over that other NBBOAgent. If the forcedtakeover flag is false, the simulator will notify the current NBBOAgent and the registration is failed.
- Upon successful registration, the simulator will generate a collection of held orders for the given class key and session name. This held order collection will be used in the EventGUIHelper. (See EventGUIHelper in simulator documents).

unregisterAgent

- always succeed

IntermarketUserSessionManager Interface

getIntermarketQuery

- Returns IntermarketQuery interface

getNBBOAgent

- Returns NBBOAgent interface

IntermarketUserAccess Interface

Logon

- NBBOAgent uses this method to log on to a CAS.

getIntermarketUserSessionManager

- NBBOAgent uses this method if the NBBOAgent has already logged in the CAS.

Notes: A detailed explanation of the NBBO Agent logon process will be provided in the CMi Programmers Guide (CMi Volume 2) – example 8.

FloorTradeMaintenanceService Interface

The simulator emulates the generation and deletion of floor trades. The methods below are used for each scenario.

acceptFloorTrade

Simulator supports the creation of floor trades on all Option products.

deleteFloorTrade

Simulator supports the deletion of floor trades on all Option products.

subscribeForFloorTradeReportsByClass

This method is used in the simulator to subscribe for floor trade reports by class.

unsubscribeForFloorTradeReportsByClass

This method is used in the simulator to unsubscribe for floor trade reports.

EventGUIHelper

Events in the EventGUIHelper

acceptNewHeldOrder

- For every one pair of session name and class key that has been registered in the simulator, the simulator finds one held order and publishes it.
- Simulator publishes the held order to registered NBBO Agent.

acceptHeldOrders

- For every one pair of session name and class key that has been registered in the simulator, the simulator finds held orders and publishes them.

acceptHeldOrderCancelReport

- For every one pair of session name and class key that has been registered in the simulator, the simulator finds one held order that has been published to the NBBO agent and creates a cancel report and publishes it.

acceptOrders (S Order)

- Simulator publishes S order to the registered NBBO Agent.

acceptSatisfactionAlert

- Simulator publishes satisfaction alert to the registered NBBO Agent.

acceptAuction

- Simulates an auction as if the user currently in the main GUI window submitted it. The user does not have to be logged in in order for this event to be published. The event will be delivered to any user subscribed for auction events.

