CONFIDENTIAL

CBOE Application Programming Interface

CBOE API Version 2.6 - Release Notes

Provides an overview of upcoming changes in the next release of the CMi

CBOE PROPRIETARY INFORMATION

31 July 2003

Document #[API-00]

Front Matter

Disclaimer

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/webAPI.

Table of Contents

Overview

The material presented in is document highlight the upcoming changes for the new release of the CMi API. The upcoming release will support CMi market linkage. Your feedback or questions regarding this document should be sent to API@cboe.com

The new interfaces will be available from the CAS server on a separate URL on the same port from the current release of the CAS server. The current CMi interfaces will still be available on the CAS in addition to the new interfaces. Following is a summary of changes by interfaces.

NBBO Agent

The NBBO agent will provide manual handling for orders. Only a DPM can become an NBBO Agent. The responsibility of the NBBO Agent could be:

Options: Manual handling of customer orders when required. For example, when an away exchange has a better price, a customer order will be removed from the book, becoming a held order, and sent to NBBO Agent for manual handling.

Stock: Manual handling of customer orders when required. For example, when an away exchange has a better price, an order will be removed from the book, becoming a held order, and sent to the NBBO Agent for manual handling. In addition, the NBBO Agent will have the ability to move any order type from the order book to manual handling at specific time before the open and up to the open for CBOE non-primary equities.

Session Management in the New CMi

The new CMi interfaces will be made available as extensions to the existing CMi on the existing CAS. A CMi user can gain access to the new interfaces by getting a reference to the IntermarketSessionManager. This reference can be obtained by one of two ways. Both options are exposed on the IntermarketUserAccess object. The IntermarketUserAccess object can be retrieved from the IOR link on the HTTP port that the CAS is publishing on.

The first option for accessing the new interface is to log into the current CMi and obtain a UserSessionManager reference. Using this reference, the CMi user can call getIntermarketUserSessionManager, which will return an IntermarketUserSessionManager interface. This interface will allow the CMi user to access the newly added methods.

The second option is to logon using the IntermarketUserAccess interface. The new logon method takes exactly the same parameter as the existing CMi's logon method and returns a newly IntermarketSessionManagerStruct. The IntermarketSessionManagerStruct contains a reference to the current CMI's UserSessionManager as well as the new CMi's IntermarketUserSessionManager.

New IDL Interfaces

CMi Interface

```
interface UserSessionManager
  {
        cmiUser::SessionProfileUserStruct getValidSessionProfileUser()
                raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException
            );
```

Currently, the MarketMaker has only one profile set as defaultProfile, which can be configured for all classes of all session or for specified classes of all session. In this release, we have added the capability for a user to have different profiles for different sessions (e.g Profile A for ONE_MAIN, Profile B for W_MAIN)

CMi Callback Interface

```
interface CMIIntermarketOrderStatusConsumer
  {
    void acceptNewHeldOrder(
      in cmiIntermarketMessages::HeldOrderDetailStruct heldOrder);
```

The NBBO agent uses this method to receive orders (S order or held order) for manual handling. The NBBOAgent can check the order origin type in the order to see if this incoming order is a S order or not.

```
    void acceptCancelHeldOrderRequest(
      in cmiProduct::ProductKeysStruct productKeys,
      in cmiIntermarketMessages::HeldOrderCancelRequestStruct cancelRequestStruct );
```

The NBBO agent uses this method to receive cancel request that originated from the customer.

```
        void acceptHeldOrderStatus(
            in cmiIntermarketMessages::HeldOrderDetailStructSequence heldOrders);
```

The NBBO agent uses this method to receive the held order status update.

```
        void acceptHeldOrderCanceledReport(
            in cmiIntermarketMessages::HeldOrderCancelReportStruct canceledReport );
```

The NBBO agent uses this method to receive reports for cancel reponses sent by the NBBO agent via IntermarketManualHandling::acceptCancelReponse

```
        void acceptHeldOrderFilledReport(
            in cmiIntermarketMessages::HeldOrderFilledReportStruct filledReport );
```

The NBBO agent uses this method to receive report after NBBO agent fills the held order.

```
        void acceptFillRejectReport(
            in cmiIntermarketMessages::OrderFillRejectStruct orderFillReject);
```

The NBBO agent use this method to receive report after NBBO agent reject a fill report via IntermarketManualHandling::rejectFill.

```
};

    interface CMINBBOAgentSessionAdmin
    {
        void acceptForcedOut(
            in string reason,
            in cmiProduct::ClassKey classKey,
            in cmiSession::TradingSessionName session );
```

The NBBO agent uses this method to receive the forced log out message when another DPM wants to take over.

```
        void acceptReminder(
            in cmiIntermarketMessages::OrderReminderStruct reminder,
            in cmiProduct::ClassKey classKey,
```

in cmiSession::TradingSessionName session);

The NBBO agent uses this method to receive reminder messages.

```
void acceptSatisfactionAlert(
  in cmiIntermarketMessages::SatisfactionAlertStruct alert,
  in cmiProduct::ClassKey classKey,
  in cmiSession::TradingSessionName session );
```

The NBBO agent uses this method to receive incoming Satisfaction Alert.

***Note**: S orders do not apply to stock.*

```
void acceptIntermarketAdminMessage(
  in cmiSession::TradingSessionName session,
  in cmiUser::Exchange originatingExchange,
  in cmiProduct::ProductKeysStruct productKeys,
  in cmiIntermarketMessages::AdminStruct adminMessage);
```

This method is for stock products only. The NBBO agent uses this method to receive admin message, this message is used for stock opening.

};

IntermarketQuery

```
interface IntermarketQuery
  {
    cmiIntermarketMessages::CurrentIntermarketStruct getIntermarketByProductForSession(
      in cmiProduct::ProductKey productKey,
      in cmiSession::TradingSessionName session )
        raises(
          exceptions::SystemException,
          exceptions::CommunicationException,
          exceptions::AuthorizationException,
          exceptions::DataValidationException,
          exceptions::NotAcceptedException
```

```
        );

    cmiIntermarketMessages::CurrentIntermarketStructSequence
getIntermarketByClassForSession(
        in cmiProduct::ClassKey classKey,
        in cmiSession::TradingSessionName session )
          raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException,
            exceptions::NotAcceptedException
          );

    cmiIntermarketMessages::AdminStructSequence getAdminMessage(
        in cmiSession::TradingSessionName session,
        in cmiProduct::ProductKey productKey,
        in cmiAdmin::MessageKey adminMessageKey,
        in cmiUser::Exchange sourceExchange)
          raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
          );

    cmiIntermarketMessages::BookDepthDetailedStruct getDetailedOrderBook(
            in cmiSession::TradingSessionName session,
            in cmiProduct::ProductKey productKey)
                  raises(
                  exceptions::SystemException,
                 exceptions::CommunicationException,
                 exceptions::AuthorizationException,
                 exceptions::DataValidationException,
                 exceptions::NotFoundException,
                 exceptions::NotAcceptedException
```

```
        );

    };
```

IntermarketManualHandling

```
void acceptSatisfactionOrderFill(
        in cmiSession::TradingSessionName session,
        in cmiOrder::OrderIdStruct satisfactionOrderId,
        in cmiOrder::OrderEntryStruct nbboAgentOrder,
        in long crowdQuantity,
        in boolean cancelRemaining,
        in cmiUtil::SatisfactionOrderDisposition disposition)
          raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException,
            exceptions::TransactionFailedException,
            exceptions::NotAcceptedException
          );
```

Incoming Satisfaction order fill- the NBBO agent submits a cross order to partially fill the S order. The NBBO agent allocates some quantity to be distributed among Market Makers. The NBBO agent also indicates if the remaining quantity should be canceled.

***Note**: S orders do not apply to stock.*

```
void acceptSatisfactionOrderInCrowdFill(
    in cmiSession::TradingSessionName session,
    in cmiProduct::ProductKey productKey,
    in cmiOrder::OrderIdStruct satisfactionOrderId,
    in long crowdQuantity,
    in boolean cancelRemaining,
    in cmiUtil::SatisfactionOrderDisposition disposition)
      raises(
        exceptions::SystemException,
        exceptions::CommunicationException,
        exceptions::AuthorizationException,
```

```
        exceptions::DataValidationException,
        exceptions::TransactionFailedException,
        exceptions::NotAcceptedException
    );
```

Incoming Satisfaction order fill- the NBBO Agent does not want to take any quantity, but rather distribute everything among Market Makers. The NBBO agent also indicates if the remaining quantity should be canceled.

***Note**: S orders do not apply to stock.*

```
void acceptSatisfactionOrderReject (
  in cmiSession::TradingSessionName session,
  in cmiProduct::ProductKey productKey,
  in cmiOrder::OrderIdStruct satisfactionOrderId,
  in cmiUtil::ActivityReason resolution)
    raises(
      exceptions::SystemException,
      exceptions::CommunicationException,
      exceptions::AuthorizationException,
      exceptions::DataValidationException,
      exceptions::TransactionFailedException,
      exceptions::NotAcceptedException
    );
```

Rejects the incoming Satisfaction order.

***Note**: S orders do not apply to stock.*

```
void acceptCustomerOrderSatisfy(
  in cmiSession::TradingSessionName session,
  in cmiOrder::OrderIdStruct referenceSatisfactonOrderId,
  in cmiOrder::OrderEntryStruct nbboAgentOrder)
    raises(
      exceptions::SystemException,
      exceptions::CommunicationException,
      exceptions::AuthorizationException,
      exceptions::DataValidationException,
      exceptions::TransactionFailedException,
      exceptions::NotAcceptedException
```

```
    );
```

Fills the customer order against the outgoing S Order.

***Note**: S orders do not apply to stock.*

```
void acceptFillReject(
        in cmiSession::TradingSessionName session,
        in cmiProduct::ProductKey productKey,
        in cmiIntermarketMessages::FillRejectRequestStruct fillRejectRequest)
          raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException,
            exceptions::TransactionFailedException,
            exceptions::NotAcceptedException
          );
```

Rejects the S order fill report for an outbound S orders.

Extension field in FillRejectRequestStruct.fillReportExtensions are:

AWAY_EXCHANGE_EXEC_ID .

EXECUTION_RECEIPT_TIME

***Note**: S orders do not apply to stock.*

```
void acceptHeldOrderReroute(
   in cmiOrder::OrderIdStruct heldOrderId,
   in cmiSession::TradingSessionName session,
   in cmiProduct::ProductKey productKey,
   in boolean nbboProtectionFlag)
      raises(
        exceptions::SystemException,
        exceptions::CommunicationException,
        exceptions::AuthorizationException,
        exceptions::DataValidationException,
```

```
        exceptions::TransactionFailedException,
        exceptions::NotAcceptedException
    );
```

The NBBO Agent reroutes held order back to the system.

```
void acceptHeldOrderByClassReroute(
    in cmiProduct::ClassKey classKey,
    in cmiSession::TradingSessionName session,
    in boolean nbboProtectionFlag)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException,
            exceptions::TransactionFailedException,
            exceptions::NotAcceptedException
        );
```

The NBBO Agent reroutes all held orders for a given class.

```
void acceptCancelResponse(
    in cmiOrder::OrderIdStruct orderId,
    in cmiUtil::CboeIdStruct cancelRequestId,
    in cmiSession::TradingSessionName session,
    in cmiProduct::ProductKey productKey)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::NotAcceptedException,
            exceptions::TransactionFailedException,
            exceptions::DataValidationException
        );
```

The NBBO Agent responds to a cancel request the originated from the customer.

```
void acceptHeldOrderFill(
   in cmiOrder::OrderIdStruct heldOrderId,
   in cmiSession::TradingSessionName session,
   in cmiOrder::OrderEntryStruct nbboAgentOrder)
      raises(
         exceptions::SystemException,
         exceptions::CommunicationException,
         exceptions::AuthorizationException,
         exceptions::NotAcceptedException,
         exceptions::TransactionFailedException,
         exceptions::DataValidationException
      );
```

The NBBO Agent fills the held order.

```
cmiIntermarketMessages::HeldOrderDetailStruct getHeldOrderById(
   in cmiSession::TradingSessionName session,
   in cmiProduct::ProductKey productKey,
   in cmiOrder::OrderIdStruct orderId)
      raises(
         exceptions::SystemException,
         exceptions::CommunicationException,
         exceptions::AuthorizationException,
         exceptions::DataValidationException,
         exceptions::NotFoundException
);
```

The NBBO Agent retrieves the held order by the given order Id.

```
cmiOrder::OrderStructSequence getAssociatedOrders(
   in cmiSession::TradingSessionName session,
   in cmiProduct::ProductKey  productKey,
   in cmiOrder::OrderIdStruct orderId)
      raises(
         exceptions::SystemException,
         exceptions::CommunicationException,
```

```
        exceptions::AuthorizationException,
        exceptions::DataValidationException
    );
```

The NBBO Agent retrieves the related PA Order for a customer order. If the orderId is of a customer order, this method will return linked PA orders or the S order. If orderId is of a Linkage Order (PA or S order) this method will return associated Customer -orders.

If no orders are found, this method will return an empty sequence.

```
cmiOrder::OrderStructSequence getOrdersByOrderTypeAndClass(
        in cmiSession::TradingSessionName session,
    in cmiProduct::ClassKey classKey,
    in cmiUser::ExchangeSequence exchanges,
    in cmiOrder::OriginTypeSequence  originTypes,
    in cmiUtil::OrderFlowDirection orderFlowDirection )
      raises(
        exceptions::SystemException,
        exceptions::CommunicationException,
        exceptions::AuthorizationException,
        exceptions::DataValidationException
      );
```

The NBBO Agent retrieves the summary report for P/A, P and S orders (inbound and outbound). (NBBO agent will be responsible to retrieve the resolution from OrderHistory). If no orders are found, an empty sequence will be returned.

```
cmiOrder::OrderStructSequence getOrdersByOrderTypeAndProduct(
    in cmiSession::TradingSessionName session,
    in cmiProduct::ProductKey productKey,
    in cmiUser::ExchangeSequence exchanges,
    in cmiOrder::OriginTypeSequence originTypes,
    in cmiUtil::OrderFlowDirection orderFlowDirection )
      raises(
        exceptions::SystemException,
        exceptions::CommunicationException,
```

```
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );
```

Same method as getOrdersByOrderTypeAndClass but orders are queried by product key.

```
    void acceptPreOpeningIndication(
      in cmiSession::TradingSessionName session,
      in cmiUser::Exchange originatingExchange,
      in cmiProduct::ProductKey productKey,
      in cmiIntermarketMessages::PreOpeningIndicationPriceStruct preOpenIndication)
        raises(
          exceptions::SystemException,
          exceptions::CommunicationException,
          exceptions::DataValidationException,
          exceptions::NotAcceptedException,
          exceptions::AuthorizationException
        );
```

Receives the pre-opening indication message. This method is used byStock ITS.

```
    void acceptPreOpeningResponse(
      in cmiSession::TradingSessionName session,
      in cmiUser::Exchange destinationExchange,
      in cmiProduct::ProductKey productKey,
      in cmiIntermarketMessages::PreOpeningResponsePriceStructSequence
preOpenResponses)
        raises(
          exceptions::SystemException,
          exceptions::CommunicationException,
          exceptions::DataValidationException,
          exceptions::NotAcceptedException,
          exceptions::AuthorizationException
        );
```

Accepts the pre-opening response message from a DPM participant in StockITS.

```
void acceptAdminMessage(
   in cmiSession::TradingSessionName session,
   in cmiUser::Exchange destinationExchange,
   in cmiProduct::ProductKey productKey,
   in cmiIntermarketMessages::AdminStruct adminMessage)
      raises(
         exceptions::SystemException,
         exceptions::CommunicationException,
         exceptions::DataValidationException,
         exceptions::NotAcceptedException,
         exceptions::AuthorizationException
      );
```

This method accepts all administrative messages used for Stock ITS.

```
void lockProduct(
   in cmiSession::TradingSessionName session,
   in cmiProduct::ProductKey productKey)
      raises(
         exceptions::SystemException,
         exceptions::CommunicationException,
         exceptions::AuthorizationException,
         exceptions::DataValidationException,
         exceptions::NotFoundException,
         exceptions::NotAcceptedException
      );
```

This method is for stock products only. It allows the NBBO Agent to lock the order book at a specified time before the open and up to the opening of a CBOE non-primary equity. This method can only be used before the open.

Not Implemented.

```
void unlockProduct(
   in cmiSession::TradingSessionName session,
   in cmiProduct::ProductKey productKey)
      raises(
         exceptions::SystemException,
         exceptions::CommunicationException,
```

```
        exceptions::AuthorizationException,
        exceptions::DataValidationException,
        exceptions::NotFoundException,
        exceptions::NotAcceptedException
    );
```

This method is for stock products only. It allows the NBBO Agent to unlock the order book and let queued orders enter the order book before the open. This is generally used if the primary opening is delayed. This is for CBOE non-primary equities that have not yet opened.

Not Implemented.

```
void rerouteBookedOrderToHeldOrder(
  in cmiOrder::OrderIdStruct bookedOrderId,
  in cmiSession::TradingSessionName session,
  in cmiProduct::ProductKey productKey,
  in boolean nbboProtectionFlag)
    raises(
      exceptions::SystemException,
      exceptions::CommunicationException,
      exceptions::AuthorizationException,
      exceptions::DataValidationException,
      exceptions::TransactionFailedException,
      exceptions::NotAcceptedException
  );
```

This method is for stock products only. It allows the NBBO Agent to remove an order in the order book and place it into manual handling. This method is available before the open of a CBOE non-primary equity.

Not Implemented.

```
void acceptOpeningPriceForProduct(
  in cmiUtil::PriceStruct openingPrice,
  in cmiSession::TradingSessionName session,
  in cmiProduct::ProductKey productKey)
    raises(
      exceptions::SystemException,
      exceptions::CommunicationException,
      exceptions::AuthorizationException,
      exceptions::DataValidationException,
```

```
        exceptions::TransactionFailedException,
        exceptions::NotAcceptedException
    );
```

This method is for stock products only. It allows the NBBO Agent to enter the opening price foran equity. Any order imbalance at this price will be routed to manual handling. This method is only available for use on CBOE non-primary equities.

Not Implemented

NBBOAgentSessionManager

```
interface NBBOAgentSessionManager
{
  IntermarketManualHandling getIntermarketManualHandling()
    raises(
      exceptions::SystemException,
      exceptions::CommunicationException,
      exceptions::AuthorizationException
    );
};
```

Provides access for the IntermarketManualHandling interface

NBBOAgent

This interfaces provides methods to register and unregister the NBBO Agent. Only a DPM user can register as an NBBO Agent. If the user is not a DPM, the system will generate an Authorization Exception.

```
NBBOAgentSessionManager registerAgent(
  in cmiProduct::ClassKey classKey,
  in cmiSession::TradingSessionName session,
  in boolean forceOverride,
  in cmiIntermarketCallback::CMIIntermarketOrderStatusConsumer imOrderStatusListener,
  in cmiIntermarketCallback::CMINBBOAgentSessionAdmin nbboAgentSessionAdmin)
    raises(
```

```
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException,
            exceptions::TransactionFailedException
        );

    void unregisterAgent(
        in cmiProduct::ClassKey classKey,
        in cmiSession::TradingSessionName session,
        in cmiIntermarketCallback::CMIIntermarketOrderStatusConsumer imOrderStatusListener,
        in cmiIntermarketCallback::CMINBBOAgentSessionAdmin nbboAgentSessionAdmin)
            raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException,
            exceptions::TransactionFailedException
        );
};
```

IntermarketUserSessionManager

IntermarketUserAccess and IntermarketUserSessionManager provide session management for the new CMI. See the “Session Management in new CMi interfaces” section in this document.

```
interface IntermarketUserSessionManager
{
    IntermarketQuery getIntermarketQuery()
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException
        );
```

```
    NBBOAgent getNBBOAgent()
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException
        );
};

struct IntermarketSessionManagerStruct
{
    cmi::UserSessionManager sessionManager;
    IntermarketUserSessionManager imSessionManager;
};
```

IntermarketUserAccess

```
interface IntermarketUserAccess
{
    IntermarketSessionManagerStruct logon(
        in cmiUser::UserLogonStruct logonStruct,
        in cmiSession::LoginSessionType sessionType,
        in cmiCallback::CMIUserSessionAdmin clientListener,
        in boolean gmdTextMessaging )
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::AuthenticationException,
                exceptions::DataValidationException
            );

    IntermarketUserSessionManager getIntermarketUserSessionManager(
        in cmi::UserSessionManager sessionManager )
```

```
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::NotFoundException
            );
        };

    };
```

NBBO Agent Order Processing and Reusing Existing Interfaces

Outgoing S Order and P/A Order

1. NBBO Agent submits a Satisfaction (S) order or P/A order using the existing OrderEntry interface. S orders do not apply to stock.

2. The NBBO Agent would use the acceptOrder method in the Order Entry interface (in the cmi.idl).

S Order

In the order entry struct:

- anOrder.orderOriginType = OrderOrigins.Satisfaction
- anOrder congingency type has to be IOC
- anOrder.extension, in this field, NBBO Agent needs to specify:
- SATISFACTION_ALERT_ID // high:low format

  This field will be used to associate an outbound S order with the satisfaction alert that the S order is based on.

- EXCHANGE_DESTINATION

P/A Order

In the order entry struct:

- anOrder.orderOriginType = OrderOrigins.Satisfaction
- anOrder congingency type has to be IOC
- anOrder.extension, in this field, NBBO Agent need to specify:
- ASSOCIATED_ORDER_ID// high:low format

  This field will be used to associate an outbound P/A order with a customer order on behalf of the P/A order.
- EXCHANGE_DESTINATION

Cancel Outbound S Orders

The NBBO agent uses the existing OrderEntry interface to cancel the outbound S order. S orders do not apply to stock.

```
void acceptCancel( in string userId, in cmiOrder::CancelRequestStruct cancelRequest, in
cmiProduct::ProductKeysStruct productKeys )
        raises(
           exceptions::SystemException,
           exceptions::CommunicationException,
           exceptions::DataValidationException,
           exceptions::TransactionFailedException,
           exceptions::NotAcceptedException,
           exceptions::AuthorizationException
        );
```

NBBO Agent Receiving Order Status

For P/A or S orders entered by the NBBO agent, the agent will get order status from the existing CMi callback CMIOrderStatusConsumer. S orders do not apply to stock.

For an order that is sent to the NBBO agent for manual handling, the NBBO agent will receive order status from the new CMi callback CMIIntermarketOrderStatusConsumer.

New Functionality Supported in the CAS Simulator

EventGUIHelper

New events added in EventGUIHelper

acceptNewHeldOrder

- Simulator publishes 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

- Simulator finds one held order that has been published to the NBBO agent and creates a cancel report and publishes it.

acceptOrder(S order)

- Simulator publishes an S order to registered NBBO Agent

acceptSatisfactionAlert

- Simulator publishes satisfaction alert to registered NBBO Agent.

Order Entry Interface

Added S order processing in simulator

acceptOrder

- If the destinationExchange is empty or CBOE, a dataValidationException with error code of DataValidationCodes.INVALID_EXCHANGE 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 order contingency type is not IOC, a DataValidationException with error code of DataValidationCodes.INVALID_CONTINGENCY_TYPE is thrown.
- If the S order quantity is greater than trade through quanitity, 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 order 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 entered by the user.

New Interfaces Supported in the CAS Simulator

IntermarketQuery Interface

getIntermarketByProductForSession

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

getIntermarketByClassForSession

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

  Note: Simulator only returns hard coded CurrentIntermarketStruct

getAdminMessage

- Not implemented in the simulator

getDetailedOrderBook

- Not implemented in the simulator

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 the simulator

acceptSatisfactionOrderInCrowdFill

- Not implemented in the simulator

acceptSatisfactionOrderReject

- Not implemented in the simulator

acceptCustomerOrderSatisfy

- Not implemented in the simulator

acceptFillReject

- Not implemented in the simulator

getAssociatedOrders

- Not implemented in the simulator

getOrdersByOrderTypeAndClass

- Not implemented in the simulator

getOrdersByOrderTypeAndProduct

- Not implemented in the simulator

acceptPreOpeningIndication

- Not implemented in the simulator

acceptPreOpeningResponse

- Not implemented in the simulator

acceptAdminMessage

- Not implemented in the simulator

acceptOpeningPriceForProduct

- Not implemented in the simulator

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 registeration, the simulator will generate a collection of held orders for the given class key and session name and publish them to NBBO Agent).

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.

Note: Example of the NBBO Agent logon process will be provided in the CMi Programmers Guide (CMi Volume 2) – example 8.