CHICAGO BOARD OPTIONS EXCHANGE

CBOE Application Programming Interface

CBOE API Version 6.0 - Release Notes

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

CBOE PROPRIETARY INFORMATION

22 May 2009

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

This document highlights upcoming changes in the new release of the CMi API, Version 6.0. Version 6.0 supports new CMi interfaces, new IDL and overall documentation changes. The sections below detail the changes in this release. Your feedback or questions regarding this document should be sent to api@cboe.com.

CMi API V6.0 Highlights

The sections below discuss upcoming CBOE*direct* software releases that introduce a new CBOE exchange, CBOE 2 (C2), and new CMi V6 interfaces. The CMi V6 interfaces support two new features; (1) the ability to participate in Directed AIM order auctions and (2) support for Market Maker Hand Held (MMHH) functionality.

CBOE 2 (C2)

The July 2009 software release of CBOE*direct* will introduce a new CBOE exchange, CBOE 2 (C2). C2 will be a fully electronic options exchange supporting a maker-taker pricing model. C2 functionality will be similar to the existing CBOE (W_MAIN) functionality with some key differences. The table below details the differences between CBOE and CBOE 2 with respect to the CMi API.

Interface	CBOE	CBOE 2	Comments
const cmiOrder::ContingencyType	Contingencies that are not accepted: -Not Held (NH)	Contingencies that are not accepted: -Minimum volume (MIN) -Market-if-Touched (MIT) -With Discretion (WD) -Midpoint Cross -Cross -Tied Cross -Autolink Cross -Autolink Cross Match -Cross_Within -Tied_Cross_Within	In C2 'Not Held' orders must be ACCEPTED where in the contingency is ignored and the orders are handled like regular orders without any contingency.
cmiSession::TradingSessionName sessionName	W_MAIN	C2_MAIN	
const cmiUser::Exchange	CBOE	CBOE2	

const BillingTypeIndicator	Billing type indicators:
	Maker=A
	Taker=R
		Billing type indicators:
		Maker=A
		Taker=R
			CBOE will be enhanced to include all the values in CBOE 2.

Interface	CBOE	CBOE 2	Comments

	Flash_Response=E
	Flash=F
	Cross=C
	Linked_AWAY=X
	Linked_AWAY_Response=L
	Opening=O
	ODD_LOT_FLASH=N
	ODD_LOT_RESPONSE=B
		Flash_Response=E
		Flash=F
		Cross=C
		Linked_AWAY=X
		Linked_AWAY_Response=L
		Opening=O
		ODD_LOT_FLASH=N
		ODD_LOT_RESPONSE=B
		RESTING=Q
		CROSS_PRICE_IMP=S
		FLASH_PRICE_IMP=T
		FLASH_RESPONSE_PRICE_IMP=U
		MAKER_TURNER=V
		RESTING_TURNER=W	

CMi V6 Functionality

Directed AIM

CBOE*direct's* Automated Improvement Mechanism (AIM) will be enhanced to include a new auction feature, Directed AIM. The Directed AIM auction will be supported in the Hybrid (W_MAIN) trading session and will be available in the July 2009 software release of CBOE*direct*.

Directed AIM gives order providers the ability to direct their orders to look for price improvement. The orders can be directed to either:

1. a target Firm (only one per order), or
2. a target DPM of a class, or
3. a target PDPM

The Directed AIM request targeted at a firm will be sent to Market Makers (MM) that are affiliated to the target firm. The Directed AIM request targeted at DPM will be sent to the DPM of the class. The Directed AIM request targeted at PDPM will be sent to the PDPM based on the PDPM assignment made in the Firm/Class routing property. If MM's should choose to price improve, they will send a matching order to start the AIM auction. The MMs also have the ability to allow the Directed AIM request to time out, or to reject the Directed AIM request.

Register for Directed AIM Auction

CMi users will have the option to choose to participate in the Directed AIM auction process via the new cmiV6 OrderQuery interface. An example of registering to participate using the OrderQuery interface would be `registerForDirectedAIM("W_MAIN",69206067).`

```
module cmiV6
        {
          interface OrderQuery : cmiV3::OrderQuery
          {
        void registerForDirectedAIM(in string sessionName, in
           cmiProduct::ClassKey classKey)
              raises(exceptions::SystemException,
                  exceptions::CommunicationException,
                  exceptions::DataValidationException,
                  exceptions::TransactionFailedException,
                  exceptions::NotAcceptedException,
                  exceptions::AuthorizationException);
          };
        }
```

The DirectedAIM indicator will be set at a Firm/Class level. The Firm should be able to configure this on a daily basis. The available values are True/False. The default value for this will be False.

- The Firms will set this value only once in a day
- Helpdesk user will have the ability to override the value if required

The selection will be removed for each firm as part of the end of day process, and reset to its default value of false. Below are examples of registration scenarios.

<u>Registration Table</u>

User Action	**Current Registration Status**	**Updated Registration For the Day**	**Result**	**Register for Direct AIM Status**
Register for Directed AIM	No Firm Affiliation	N/A	Data validation code: USER_NOT_AFFILIATED_TO_ANY_FIRM =7005	No.
Register for Directed AIM	Registered	No	Data validation code: ALREADY_UPDATES_AS_REGISTERED = 7007	Yes

User Action	Current Registration Status	Updated Registration For the Day	Result	Register for Direct AIM Status
Register for Directed AIM	Registered	Yes	Data validation code: ALREADY_UPDATES_AS_REGISTERED = 7007	Yes
Register for Directed AIM	Unregistered	No	Register	Yes

Order Submission

As with all auctions, both a primary order and a match order must be submitted for auction participation.

Primary Order

The extensions field in the orderEntryStruct will be enhanced to allow users to send a Directed AIM request to a target firm/PMM or DPM. Valid values for the extensions field are:

- dfirm = Affiliated Firm Acronym if the user wants to target a Firm.
- dfirm = DDPM if the user wants to target the DPM of the class.
- dfirm = DPMM if the user wants to target the PDPM for the order provider.

The optionalData field will contain the value, A:AIR;

Example:

- **dfirm=PAX ->** where PAX is the Affiliated Firm Acronym
- **dfirm=DDPM ->** where DDPM indicates DPM choice
- **dfirm=DPMM ->** where DPMM indicates PDPM choice

Match Order

The Directed AIM Response Order from the Firm (Match Order) will include the primary order information in the optionalData field as **DAIM:highcobeid:lowcboeid**; Where **highcobeid:lowcboeid** is provided in the Directed AIM notification message published out to the users.

```
module cmiOrder
        struct OrderEntryStruct
           {
```

```
        cmiUser::ExchangeFirmStruct executingOrGiveUpFirm;
        string branch;
        long branchSequenceNumber;
        string correspondentFirm;
        string orderDate; // YYYYMMDD format
        cmiUser::ExchangeAcronymStruct originator;
        long originalQuantity;
        cmiProduct::ProductKey productKey;
        cmiUtil::Side side;
        cmiUtil::PriceStruct price;
        cmiOrder::TimeInForce timeInForce;
        cmiUtil::DateTimeStruct expireTime;
        cmiOrder::OrderContingencyStruct contingency;
        cmiUser::ExchangeFirmStruct cmta;
        string extensions;
         string account;
        string subaccount;
        cmiOrder::PositionEffect positionEffect;
        cmiOrder::CrossingIndicator cross;
        cmiOrder::OriginType orderOriginType;
        cmiOrder::Coverage coverage;
        cmiOrder::NBBOProtectionType orderNBBOProtectionType;
        string optionalData;
        string userAssignedId;
        cmiSession::TradingSessionNameSequence sessionNames;
    };
    typedef sequence <OrderEntryStruct> OrderEntryStructSequence;

module cmiConstants
        interface ExtensionFields{

        // This extension is added for Directed AIM
                const ExtensionField DIRECTED_FIRM = "dfirm";
                const ExtensionField DPM = "DDPM";
                const ExtensionField PDPM = "DPMM";
                }
```

Error Codes and Constants

Below are the new CMi constants and error codes that correspond to Directed AIM.

```
module cmiConstants
        {
        interface ActivityTypes{
                // DirectedAIM Notification Start and End
                const cmiTraderActivity::ActivityType DIRECTED_AIM_NOTIFICATION_START =
                817;
                const cmiTraderActivity::ActivityType DIRECTED_AIM_NOTIFICATION_END =
                818;
                        }

        interface AuctionTypes{
                // Auction type codes
                const cmiOrder::AuctionType AUCTION_DAIM = 8;
                        }
        }
module cmiErrorCodes
{
          interface DataValidationCodes  {

        // DirectedAIM NotAccepted Codes.
        const exceptions::ErrorCode DIRECTED_AIM_PRIMARY_EXPIRED = 7000;
        const exceptions::ErrorCode NOT_REGISTERED_FOR_DIRECTED_AIM = 7001;
        const exceptions::ErrorCode NO_PDPM_AVAILABLE_FOR_DIRECTED_AIM = 7002;
        const exceptions::ErrorCode NO_DPM_AVAILABLE_FOR_DIRECTED_AIM = 7003;
        const exceptions::ErrorCode NO_AFFILIATED_MM_AVAILABLE_FOR_DIRECTED_AIM =
        7004;
        const exceptions::ErrorCode USER_NOT_AFFILIATED_TO_ANY_FIRM = 7005;
        const exceptions::ErrorCode INVALID_AFFILIATED_FIRM = 7006;
        const exceptions::ErrorCode ALREADY_UPDATED_AS_REGISTERED = 7007;
        const exceptions::ErrorCode ALREADY_UPDATED_AS_UNREGISTERED = 7008;
                }
}
```

Examples of Data Validation Errors

The table below gives examples of data validation scenarios.

No.	Action	Expect Data Validation Exception Codes
1.	When Directed AIM Notification has expired or the 1st match Order is received.	`DIRECTED_AIM_PRIMARY_EXPIRED`
2.	When Directed AIM order is sent to the targeted firm, but the target firm is not registered for Directed AIM.	`NOT_REGISTERED_FOR_DIRECTED_AIM`
3.	When the target is DPMM => PDPM choice, but there are no PMMs setup in routing property.	`NO_PDPM_AVAILABLE_FOR_DIRECTED_AIM`
4.	When the target is DDPM =>choice, but there is no DPM setup for the class.	`NO_DPM_AVAILABLE_FOR_DIRECTED_AIM`
5.	When target firm exists but there are no affiliated MMs for the Firm, the Directed AIM order will get rejected.	`NO_AFFILIATED_MM_AVAILABLE_FOR_DIRECTED_AIM`
6.	If a DPM User is not affiliated to any firm and Directed AIM is targeted for the DPM user.	`USER_NOT_AFFILIATED_TO_ANY_FIRM`
7.	When user tries to register more than once. But user subscription for auction type Directed AIM does not fail.	`ALREADY_UPDATED_AS_REGISTERED`
8.	Whenever user tries to unregister more than once. User will not unsubscribe for auction types Directed AIM for that specific class.	`ALREADY_UPDATED_AS_UNREGISTERED`

Cancel/Cancel Replace Request for Directed AIM Order

Users will be allowed to cancel and cancel replace their Directed AIM request. However, if the targeted Firm has responded to the Directed AIM request and the Directed AIM auction has started, then the cancel or cancel replace request will wait until the end of the auction. For cancel replace of a Directed AIM order, the orderEntry struct will include dfirm in the extensions field and primary order information in the optional data field.

Market Maker Hand Held to CBOEdirect

The August 2009 software release of CBOE*direct* will encompass Market Maker Hand Held (MMHH) functionality. The existing Market Maker Hand Held system gives Firm traders the ability to electronically enter their trades on a firm's handheld device and send them for processing to the MMHH System via the NCC interface. With the new CMi V6 interfaces, CMi users will have the ability to perform MMHH functionality using the CBOE*direct* platform instead of the NCC interface.

Login to CBOEdirect using the UserAccessV6 Interface

In order to perform a MMHH trade on CBOE*direct*, users must login using the UserAccessV6 interface. The UserAccessV6 interface references the UserSessionManagerV6, which references the FloorTradeMaintenanceService. The FloorTradeMaintenanceService is used to add, delete, subscribe and unsubscribe for MMHH trades.

```
    interface UserAccessV6
    {
            UserSessionManagerV6 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,
              exceptions::NotFoundException
          );
   };
   interface UserSessionManagerV6 : cmiV5::UserSessionManagerV5
   {
            FloorTradeMaintenanceService getFloorTradeMaintenanceService()
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException
        );
   };
```

Generate a Market Maker Hand Held Trade

A MMHH trade is generated by calling the acceptFloorTrade method in the cmiV6:FloorTradeMaintenanceService interface. The acceptFloorTrade method takes the cmiTrade:FloorTradeEntryStruct as an argument and returns the cmiUtil:CboeIdStruct on successful trade generation. Considerations for generating a MMHH trade are:

- Only Market Makers (i.e. MarketMaker (M) and DPM_Role (D)) are allowed to submit a MMHH trade.
- If a MMHH trade is not generated successfully, the API will throw an exception specifying the general cause.
- The Market Maker entering the trade will not receive trade fill reports. Successful return of the method call should be considered as the confirmation.
- The CBOE*direct* trade engine has specific validation rules for allowing MMHH trades based on userId, executingAcronym, sessionName and ProductKey combinations. It maintains an internal mapping table for userId/acronym combinations. This table is configurable by the CBOE Help Desk. Based on passing parameters, the trade engine performs two look-ups in order to obtain the user information to carry on the trade. The first look-up looks for a match between the acronym corresponding to the user profile derived from the passed user id and the passed executing acronym. If this look-up does not return positive results then a second look-up is performed. The second look-up looks for an association between the executing acronym and the passed user id. Such association is entered and maintained as a firm property. If the second look-up does not find a match then the trade cannot proceed and an error message explaining the problem is sent as an exception.

```
interface FloorTradeMaintenanceService
{
        cmiUtil::CboeIdStruct acceptFloorTrade
                        (
                        in cmiTrade::FloorTradeEntryStruct floorTrade)
            raises(
              exceptions::SystemException,
              exceptions::CommunicationException,
              exceptions::AuthorizationException,
              exceptions::DataValidationException,
              exceptions::NotAcceptedException,
                     exceptions::TransactionFailedException
             );
 module cmiTrade
 {
        struct FloorTradeEntryStruct
```

```
{
                cmiSession::TradingSessionName sessionName;
cmiProduct::ProductKey productKey;
                long quantity;
                cmiUtil::PriceStruct price;
                cmiUtil::Side side;
                string account;
       string subaccount;
cmiUser::ExchangeFirmStruct cmta;
                cmiUser::ExchangeAcronymStruct executingMarketMaker;
                cmiUser::ExchangeFirmStruct firm;
                char positionEffect;
                cmiUser::ExchangeAcronymStruct contraBroker;
                cmiUser::ExchangeFirmStruct contraFirm;
                cmiUtil::DateTimeStruct timeTraded;
                string optionalData;
};
```

The following validations apply to the cmiTrade::FloorTradeEntryStruct.

Input Field Name	**Mandatory or Optional Input**	**Valid Format Check**	**Valid Range Check**	**Existence Check**
Trading Session Name	Mandatory			√
Product Key	Mandatory			√
Quantity	Mandatory	N/A	√	
Price	Mandatory	N/A	√	
Side	Mandatory			√
Executing Broker Acronym	Mandatory	√		√
Contra Broker	Mandatory	√		
Contra Firm	Mandatory	√		
Position Effect	Optional			√
Account	Optional			
Subaccount	Optional			
CMTA (Exchange + Firm)	Optional		√ **(Firm)**	√ **(Exchange)**
Executing Firm	Optional			
Optional Data	Optional			

Table Legend

N/A: Not Applicable

√ : Validated

Blank cell: Field is not validated for that specific check

Delete a Market Maker Hand Held Trade

The cmiV6:FloorTradeMaintenanceService is used to delete MMHH trades. A valid user, tradeId, tradingSession and productKey must be specified to delete the trade. ExchangeFirm is an optional field for deletion. If a trade is not deleted successfully an exception will be thrown specifying the general cause. Trade Bust reports will not be sent for deleted trades.

```
interface FloorTradeMaintenanceService
{
        void deleteFloorTrade
                        (
                in cmiSession::TradingSessionName sessionName,
                in cmiProduct::ProductKey productKey,
                in cmiUtil::CboeIdStruct tradeId,
                in cmiUser::ExchangeAcronymStruct user,
                in cmiUser::ExchangeFirmStruct firm,
                in string reason)
        raises(
           exceptions::SystemException,
           exceptions::CommunicationException,
           exceptions::AuthorizationException,
           exceptions::DataValidationException,
           exceptions::NotAcceptedException,
           exceptions::NotFoundException,
           exceptions::TransactionFailedException
         );
```

Subscribe for a Market Maker Hand Held Trade

Whenever a PAR trade is received by the Order Handling Service (OHS) it generates a CMi based quote fill message (cmiQuote::QuoteFilledReportStruct). This message will be used for both Market Maker Trade Notifications (MMTN) and Floor Trade Reports.

- In order to distinguish between a regular quote fill message and a MMTN, the QuoteId is always set to 0 for MMTN.
- A user can subscribe for MMHH notifications using the cmiV6::FloorTradeMaintenanceService as shown below. If the classKey is zero, the user receives MMHH trade notifications for all classes. If the classKey has a valid non-zero value, the user gets MMHH trade notifications for the specified classKey only. If neither subscription exists, the CAS will drop the subscription.
- A user can specify either the same consumer or different consumers for their subscriptions. The CAS does not prevent users from using their regular QuoteStatus consumer as the FloorTradeReport consumer. The transaction sequence number in the report could be any number so assumptions should not be made on it.
- CBOE Help Desk has to turn on a specific property known as the “Firm Market Maker Trade Notification Parameter" in order to activate trade notifications. By default it is turned off. Firms have to contact the CBOE Help Desk for activation.
- CAS subscription settings do not go to the server. So, in order to have a user receive the MMTNs successfully, user should be setup correctly in both the places. Both side setups are independent and different. So, it is the user’s responsibility to send the correct subscriptions to the CAS.

```
interface FloorTradeMaintenanceService

    void subscribeForFloorTradeReportsByClass
            (
                    in cmiCallbackV2::
                    CMIQuoteStatusConsumer floorTradeReportConsumer,
                    in cmiProduct::ClassKey classKey,
                    in boolean gmdCallback)
          raises(
             exceptions::SystemException,
             exceptions::CommunicationException,
             exceptions::AuthorizationException,
             exceptions::DataValidationException
            );
```

Unsubscribe for a Market Maker Hand Held Trade

Users can turn off MMHH trade notifications by using the unsubscribeForFloorTradeReportsByClass method. If the classKey is zero, notification is turned off for all classes. If the classKey has a valid non-zero value, notification is turned off only for the specified classKey.

```
        interface FloorTradeMaintenanceService

                void unsubscribeForFloorTradeReportsByClass
                        (
                                in cmiCallbackV2::CMIQuoteStatusConsumer
                                        floorTradeReportConsumer,
                                in cmiProduct::ClassKey classKey)
                                raises(
                                exceptions::SystemException,
                               exceptions::CommunicationException,
                               exceptions::AuthorizationException,
                               exceptions::DataValidationException
                                );
};
```

IDL Interfaces

New and modified IDL is reflected in **bold** face.

```
module cmiV6
{
        interface OrderQuery : cmiV3::OrderQuery
  {
        void registerForDirectedAIM(in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKey classKey)
         raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::DataValidationException,
            exceptions::TransactionFailedException,
            exceptions::NotAcceptedException,
            exceptions::AuthorizationException
         );
```

```
   };
            interface FloorTradeMaintenanceService
   {
      cmiUtil::CboeIdStruct acceptFloorTrade(in cmiTrade::FloorTradeEntryStruct floorTrade)
           raises(
              exceptions::SystemException,
              exceptions::CommunicationException,
              exceptions::AuthorizationException,
              exceptions::DataValidationException,
              exceptions::NotAcceptedException,
                      exceptions::TransactionFailedException
           );

      void deleteFloorTrade(       in cmiSession::TradingSessionName sessionName,
                                   in cmiProduct::ProductKey productKey,
                                   in cmiUtil::CboeIdStruct tradeId,
                                   in cmiUser::ExchangeAcronymStruct user,
                                   in cmiUser::ExchangeFirmStruct firm,
                                   in string reason
                         )
           raises(
              exceptions::SystemException,
              exceptions::CommunicationException,
              exceptions::AuthorizationException,
              exceptions::DataValidationException,
              exceptions::NotAcceptedException,
              exceptions::NotFoundException,
              exceptions::TransactionFailedException
           );

      void subscribeForFloorTradeReportsByClass(in cmiCallbackV2::CMIQuoteStatusConsumer
floorTradeReportConsumer, in cmiProduct::ClassKey classKey, in boolean gmdCallback)
           raises(
              exceptions::SystemException,
              exceptions::CommunicationException,
              exceptions::AuthorizationException,
                exceptions::DataValidationException
                        );

        void unsubscribeForFloorTradeReportsByClass(in
cmiCallbackV2::CMIQuoteStatusConsumer floorTradeReportConsumer, in cmiProduct::ClassKey
classKey)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
                        );

    };

    interface UserSessionManagerV6 : cmiV5::UserSessionManagerV5  {
            cmiV6::OrderQuery   getOrderQueryV6()
          raises(
           exceptions::SystemException,
           exceptions::CommunicationException,
           exceptions::AuthorizationException
          );

        FloorTradeMaintenanceService getFloorTradeMaintenanceService()
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException
            );
    };

    interface UserAccessV6
    {
        UserSessionManagerV6 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,
                exceptions::NotFoundException
            );

    };



   interface ProductQueryV6: cmi::ProductQuery
   {
      cmiProduct::ProductGroup getProductGroups()
           raises(
                        exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::DataValidationException,
                exceptions::NotFoundException,
                exceptions::AuthorizationException );
      cmiProduct::ClassKeySequence getClassesForProductGroup(in cmiProduct::ProductGroup
productGroupName)
                    raises(
                              exceptions::SystemException,
                              exceptions::CommunicationException,
                              exceptions::DataValidationException,
                     exceptions::NotFoundException,
                              exceptions::AuthorizationException );
   };
        };
```

module cmiConstants

{

        interface ExtensionFields

```
{
   // Used for routing an order to a BART terminal
   const ExtensionField BARTID = "BARTID";
   // Firm information for stock leg of a buy-write
   const ExtensionField STOCK_FIRM = "STOCK_FIRM";
   const ExtensionField STOCK_FIRM_NAME = "STOCK_FIRM_NAME";
   const ExtensionField MEET_LOCATION_IN = "9380";
   const ExtensionField MEET_FIRM_NAME = "9381";
   const ExtensionField MEET_LOCATION_OUT = "207";
        // The following are used for linkage
   const ExtensionField CBOE_EXEC_ID = "cboeExecId";
   const ExtensionField ORIGINAL_QUANTITY = "originalQuantity";
   const ExtensionField SIDE = "side";
   const ExtensionField EXEC_BROKER = "execBroker";
   const ExtensionField ORS_ID = "orsId";
   const ExtensionField SATISFACTION_ALERT_ID = "satAlertId";
   const ExtensionField ASSOCIATED_ORDER_ID = "assocOrderId";
   const ExtensionField LINKAGE_MECHANISM = "LinkageMechanism";
   const ExtensionField EXPIRATION_TIME = "ExpirationTime";
   const ExtensionField ORIGINAL_ORDER_ACRONYM = "originalOrderUserAcronym";
   const ExtensionField BROKER_ROUTING_ID = "6818";

   const ExtensionField AWAY_CANCEL_REPORT_EXEC_ID="awayCancelReportExecId";
   const ExtensionField AWAY_EXCHANGE_USER_ACRONYM="1";
   const ExtensionField USER_ASSIGNED_CANCEL_ID="11";
   const ExtensionField AWAY_EXCHANGE_EXEC_ID="17";
   const ExtensionField HANDLING_INSTRUCTION="21";
   const ExtensionField AWAY_EXCHANGE_ORDER_ID = "37";
   const ExtensionField TEXT = "58";
   const ExtensionField AWAY_EXCHANGE_TRANSACT_TIME = "60";
   const ExtensionField EXCHANGE_DESTINATION = "100";
   const ExtensionField AUTO_EXECUTION_SIZE = "5201";
   const ExtensionField TRADE_THRU_TIME = "5202";
   const ExtensionField TRADE_THRU_SIZE = "5203";
   const ExtensionField TRADE_THRU_PRICE = "5204";
   const ExtensionField ADJUSTED_PRICE_INDICATOR = "5205";
```

```
    const ExtensionField SATISFACTION_ORDER_DISPOSITION = "5206";
    const ExtensionField EXECUTION_RECEIPT_TIME = "5207";
    const ExtensionField ORIGINAL_ORDER_TIME = "5208";
    const ExtensionField OLA_REJECT_REASON = "5209";
    const ExtensionField ORDER_CAPACITY = "6528";
    const ExtensionField ORDER_RESTRICTIONS = "6529";

    // The following are used for Auction response
    const ExtensionField AUCTION_ID = "auctionId";
    const ExtensionField BILLING_TYPE = "billingType"; // CBSX Billing Enhancements

    // The following are used for TradeThrough Processing
    const ExtensionField FADE_EXCHANGE = "FADE_EXCHANGE";

   // The following extensions field added for COB Auction Message Change
    const ExtensionField EXECUTING_FIRM = "firm";
    const ExtensionField CORRESPONDENT_FIRM = "corresfirm";
    const ExtensionField CMTA_FIRM = "cmta";
    const ExtensionField NBBO_BID_PRICE = "nbbobid";
    const ExtensionField NBBO_ASK_PRICE = "nbboask";

   // The following is used by the GUI
    const ExtensionField GUICFI  = "guicfi";

    // This extension is added for Directed AIM
    const ExtensionField DIRECTED_FIRM = "dfirm";
    const ExtensionField DPM = "DDPM";
    const ExtensionField PDPM = "DPMM";
  };


interface ExchangeStrings
   {
   const cmiUser::Exchange AMEX    = "AMEX";   //American Stock Exchange
   const cmiUser::Exchange BSE     = "BSE";    //Boston Stock Exchange
   const cmiUser::Exchange CBOE    = "CBOE";   //Chicago Board Options Exchange
   const cmiUser::Exchange CBOE2   = "CBOE2";  //Chicago Board Options Exchange 2
   const cmiUser::Exchange CBOT    = "CBOT";   //Chicago Board of Trade
   const cmiUser::Exchange CHX     = "CHX";    //Chicago Stock Exchange
   const cmiUser::Exchange CME     = "CME";    //Chicago Mercantile Exchange
```

```
    const cmiUser::Exchange CSE     = "CSE";    //Cincinnati Stock Exchange
    const cmiUser::Exchange ISE     = "ISE";    //International Stock Exchange
    const cmiUser::Exchange LIFFE   = "LIFFE";  //International Financial Futures and Options
Exchange
    const cmiUser::Exchange NASD    = "NASD";   //National Association of Securities Dealers
    const cmiUser::Exchange NYME    = "NYME";   //New York Mercantile Exchange
    const cmiUser::Exchange NYSE    = "NYSE";   //New York Stock Exchange
    const cmiUser::Exchange ONE     = "ONE";    //OneChicago Exchange
    const cmiUser::Exchange PHLX    = "PHLX";   //Philadelphia Stock Exchange
    const cmiUser::Exchange PSE     = "PSE";    //Pacific Stock Exchange
    const cmiUser::Exchange NQLX    = "NQLX";   //Nasdaq Liffe Markets
    const cmiUser::Exchange BOX     = "BOX";    // Boston Options Exchange
    const cmiUser::Exchange CFE     = "CFE";    //CBOE Futures Exchange
    const cmiUser::Exchange NSX     = "NSX";    //National Stock Exchange
    const cmiUser::Exchange NASDAQ  = "NASDQ";  //National Association of Securities Dealers
Automated Quotation
    const cmiUser::Exchange BATS    = "BATS";   //Better Alternative Trading System
    };


interface BillingTypeIndicators
    {
        const BillingTypeIndicator MAKER                  = 'A';
        const BillingTypeIndicator TAKER                  = 'R';
        const BillingTypeIndicator FLASH_RESPONSE         = 'E';
        const BillingTypeIndicator FLASH                  = 'F';
        const BillingTypeIndicator CROSS                  = 'C';
        const BillingTypeIndicator LINKED_AWAY            = 'X';
        const BillingTypeIndicator LINKED_AWAY_RESPONSE = 'L';
        const BillingTypeIndicator OPENING                = 'O';
        const BillingTypeIndicator ODD_LOT_FLASH          = 'N';
        const BillingTypeIndicator ODD_LOT_RESPONSE       = 'B';
       const BillingTypeIndicator RESTING                 = 'Q';
       const BillingTypeIndicator CROSS_PRICE_IMP         = 'S';
       const BillingTypeIndicator FLASH_PRICE_IMP         = 'T';
       const BillingTypeIndicator FLASH_RESPONSE_PRICE_IMP = 'U';
       const BillingTypeIndicator MAKER_TURNER                    = 'V';
       const BillingTypeIndicator RESTING_TURNER          = 'W';

    };


  interface ActivityTypes

     {

        // Order Activity Events

        const cmiTraderActivity::ActivityType  NEW_ORDER              = 1;

        const cmiTraderActivity::ActivityType  FILL_ORDER             = 2;

        const cmiTraderActivity::ActivityType  CANCEL_ORDER           = 3;

        const cmiTraderActivity::ActivityType  BUST_ORDER_FILL        = 4;

        const cmiTraderActivity::ActivityType  BUST_REINSTATE_ORDER    = 5;

        const cmiTraderActivity::ActivityType  CANCEL_REPLACE_ORDER    = 6;
        const cmiTraderActivity::ActivityType  UPDATE_ORDER           = 7;
        const cmiTraderActivity::ActivityType  BOOK_ORDER             = 8;
        const cmiTraderActivity::ActivityType  STATE_CHANGE_ORDER     = 9;
        const cmiTraderActivity::ActivityType  PRICE_ADJUST_ORDER     = 10;
        const cmiTraderActivity::ActivityType  CANCEL_ALL_ORDERS      = 11;
        const cmiTraderActivity::ActivityType  HELD_FOR_IPP_PROTECTION = 12;       // new
for IPP
        const cmiTraderActivity::ActivityType  CANCEL_REPLACE_ORDER_REQUEST = 13;
        const cmiTraderActivity::ActivityType  ORDER_ROUTED = 14;
        const cmiTraderActivity::ActivityType  CROSSING_ORDER_ROUTED = 15;
        const cmiTraderActivity::ActivityType  FAILED_ROUTE                    = 16;
        const cmiTraderActivity::ActivityType  CANCEL_REQUEST_ROUTED   = 17;
            const cmiTraderActivity::ActivityType  CANCEL_REQUEST_FAILED_ROUTE = 18;
        const cmiTraderActivity::ActivityType
CANCEL_REPLACE_ORDER_REQUEST_FAILED_ROUTE = 19 ;


        // Strategy Order leg activity types
        const cmiTraderActivity::ActivityType  NEW_ORDER_STRATEGY_LEG     = 51;
        const cmiTraderActivity::ActivityType  FILL_STRATEGY_LEG          = 52;
        const cmiTraderActivity::ActivityType  CANCEL_STRATEGY_LEG        = 53;
        const cmiTraderActivity::ActivityType  BUST_STRATEGY_LEG_FILL     = 54;
        const cmiTraderActivity::ActivityType  BUST_REINSTATE_STRATEGY_LEG = 55;
        const cmiTraderActivity::ActivityType  BOOK_STRATEGY_LEG          = 56;
        const cmiTraderActivity::ActivityType  UPDATE_STRATEGY_LEG        = 57;
        const cmiTraderActivity::ActivityType  PRICE_ADJUST_ORDER_LEG     = 60;
        const cmiTraderActivity::ActivityType  MANUAL_TA_TIMEOUT_STRATEGY_LEG =
70;
        const cmiTraderActivity::ActivityType  MANUAL_BOOK_TIMEOUT_STRATEGY_LEG
= 71;
        const cmiTraderActivity::ActivityType
MANUAL_AUCTION_TIMEOUT_STRATEGY_LEG = 72;
        const cmiTraderActivity::ActivityType  MANUAL_FILL_TIMEOUT_STRATEGY_LEG =
73;
        const cmiTraderActivity::ActivityType  MANUAL_FILL_REJECT_STRATEGY_LEG =
74;


        // Quote Activity Events
        const cmiTraderActivity::ActivityType  NEW_QUOTE             = 101;
```

```
    const cmiTraderActivity::ActivityType  FILL_QUOTE            = 102;
    const cmiTraderActivity::ActivityType  CANCEL_QUOTE          = 103;
    const cmiTraderActivity::ActivityType  CANCEL_ALL_QUOTES     = 104;
    const cmiTraderActivity::ActivityType  SYSTEM_CANCEL_QUOTE   = 105;
    const cmiTraderActivity::ActivityType  UPDATE_QUOTE          = 106;
    const cmiTraderActivity::ActivityType  BUST_QUOTE_FILL       = 107;

    // Strategy Quote leg activity types
    const cmiTraderActivity::ActivityType  QUOTE_LEG_FILL        = 152;
    const cmiTraderActivity::ActivityType  BUST_QUOTE_LEG_FILL   = 157;

    // RFQ Activity Events
    const cmiTraderActivity::ActivityType  NEW_RFQ               = 201;

    // New Activity Types for Linkage
    const cmiTraderActivity::ActivityType INBOUND_S_ORDER_FILL
      = 300;
    const cmiTraderActivity::ActivityType NEW_ORDER_REJECT = 301;
    const cmiTraderActivity::ActivityType FILL_REJECT = 302;
    const cmiTraderActivity::ActivityType CANCEL_ORDER_REQUEST = 303;
    const cmiTraderActivity::ActivityType CANCEL_ORDER_REQUEST_REJECT = 304;
    const cmiTraderActivity::ActivityType CANCEL_REPORT_REJECT = 305;
    const cmiTraderActivity::ActivityType NEW_ORDER_REJECT_REJECTED = 306;
    const cmiTraderActivity::ActivityType FILL_REJECT_REJECTED = 307;
    const cmiTraderActivity::ActivityType
CANCEL_ORDER_REQUEST_REJECT_REJECTED = 308;
    const cmiTraderActivity::ActivityType CANCEL_REPORT_REJECT_REJECTED = 309;
    const cmiTraderActivity::ActivityType ROUTE_TO_AWAY_EXCHANGE = 310;
    const cmiTraderActivity::ActivityType LINKAGE_ORDER_RELATIONSHIP
      = 311;
      const cmiTraderActivity::ActivityType
EXECUTION_REPORT_ON_LINKED_ORDER                 = 312;
      const cmiTraderActivity::ActivityType EXECUTION_REPORT_ROUTED
      = 313;
      const cmiTraderActivity::ActivityType EXECUTION_REPORT_FAILED_ROUTE
      = 314;
      const cmiTraderActivity::ActivityType AWAY_EXCHANGE_MARKET          = 315;
```

```
        const cmiTraderActivity::ActivityType LINKAGE_DISQUALIFIED_EXCHANGE = 
316;


    //New Activity Types for Auction
    const cmiTraderActivity::ActivityType AUCTION_START = 401;
    const cmiTraderActivity::ActivityType AUCTION_TRIGGER_START = 402;
    const cmiTraderActivity::ActivityType AUCTION_END = 403;
    const cmiTraderActivity::ActivityType AUCTION_TRIGGER_END = 404;

    //TSB Request events
    const cmiTraderActivity::ActivityType TSB_REQUEST = 501;

    //Volume maintenance event
    const cmiTraderActivity::ActivityType VOL_MAINTENANCE                  = 601;
    //Par Broker Select Time event
    const cmiTraderActivity::ActivityType PAR_BROKER_USED_MKT_DATA         = 602;
    const cmiTraderActivity::ActivityType PAR_BROKER_MKT_DATA              = 603;
    const cmiTraderActivity::ActivityType PAR_BROKER_LEG_MKT               = 604;

            //ManualOrder return events
    const cmiTraderActivity::ActivityType MANUAL_ORDER_TA          = 701;
    const cmiTraderActivity::ActivityType MANUAL_ORDER_TB          = 702;
    const cmiTraderActivity::ActivityType MANUAL_ORDER_BOOK        = 703;
    const cmiTraderActivity::ActivityType MANUAL_ORDER_AUCTION     = 704;

    //PAR print activity events
    const cmiTraderActivity::ActivityType PAR_PRINT_INTRA_DAY      = 705;
    const cmiTraderActivity::ActivityType PAR_PRINT_END_OF_DAY     = 706;
    const cmiTraderActivity::ActivityType MANUAL_FILL_REJECT       = 707;

    const cmiTraderActivity::ActivityType MANUAL_ORDER_TA_TIMEOUT          = 708;
    const cmiTraderActivity::ActivityType MANUAL_ORDER_TB_TIMEOUT          = 709;
    const cmiTraderActivity::ActivityType MANUAL_ORDER_BOOK_TIMEOUT        = 710;
    const cmiTraderActivity::ActivityType MANUAL_ORDER_AUCTION_TIMEOUT
        = 711;
    const cmiTraderActivity::ActivityType MANUAL_ORDER_LINKAGE_TIMEOUT
        = 712;
```

```
const cmiTraderActivity::ActivityType MANUAL_FILL_TIMEOUT = 713;
const cmiTraderActivity::ActivityType MANUAL_FILL_LINKAGE_TIMEOUT = 714;
const cmiTraderActivity::ActivityType MANUAL_ORDER_REROUTE_REQUEST = 715;
const cmiTraderActivity::ActivityType MANUAL_ORDER_REROUTE_CROWD_REQUEST = 716;
const cmiTraderActivity::ActivityType FORCED_LOGOFF_PAR = 717;

// ManualTimeouts Failure Events. 800 series.
const cmiTraderActivity::ActivityType MANUAL_FILL_REJECT_FAILURE = 807;
const cmiTraderActivity::ActivityType MANUAL_ORDER_TA_TIMEOUT_FAILURE = 808;
const cmiTraderActivity::ActivityType MANUAL_ORDER_TB_TIMEOUT_FAILURE = 809;
const cmiTraderActivity::ActivityType MANUAL_ORDER_BOOK_TIMEOUT_FAILURE = 810;
const cmiTraderActivity::ActivityType MANUAL_ORDER_AUCTION_TIMEOUT_FAILURE = 811;
const cmiTraderActivity::ActivityType MANUAL_FILL_TIMEOUT_FAILURE = 813;
const cmiTraderActivity::ActivityType MANUAL_FILL_LINKAGE_TIMEOUT_FAILURE = 814;
const cmiTraderActivity::ActivityType FORCED_LOGOFF_PAR_FAILURE = 815;

// Non-order Message Reroute Events
const cmiTraderActivity::ActivityType NON_ORDER_MESSAGE_REROUTE = 901;

// AUDIT History Event
const cmiTraderActivity::ActivityType AUDIT_HISTORY_EVENT = -100;

//Added for New Options Linkage Sweep and Return Functionality
const cmiTraderActivity::ActivityType MANUAL_ORDER_SR = 718;
const cmiTraderActivity::ActivityType MANUAL_ORDER_SR_TIMEOUT= 719;
const cmiTraderActivity::ActivityType MANUAL_ORDER_SR_TIMEOUT_FAILURE=816;

// DirectedAIM Notification Start and End
```

const cmiTraderActivity::ActivityType DIRECTED_AIM_NOTIFICATION_START = 817;

const cmiTraderActivity::ActivityType DIRECTED_AIM_NOTIFICATION_END = 818;

```
interface AuctionTypes
        {
                const cmiOrder::AuctionType AUCTION_INTERNALIZATION =1;
          const cmiOrder::AuctionType AUCTION_STRATEGY =2;
          const cmiOrder::AuctionType AUCTION_REGULAR_SINGLE =3;
          const cmiOrder::AuctionType AUCTION_HAL = 4;
          const cmiOrder::AuctionType AUCTION_SAL = 5;
          const cmiOrder::AuctionType AUCTION_UNSPECIFIED = 0;
          // sharing the Auction channel
          const cmiOrder::AuctionType STOCK_NBBO_FLASH = 6;
          const cmiOrder::AuctionType STOCK_ODD_LOT = 7;
          const cmiOrder::AuctionType AUCTION_DAIM = 8;
        };
```

module cmiErrorCodes

```
    interface NotAcceptedCodes  {
        const exceptions::ErrorCode UNKNOWN_TYPE = 4000;
        const exceptions::ErrorCode INVALID_STATE = 4010;
        const exceptions::ErrorCode INVALID_REQUEST = 4020;
        const exceptions::ErrorCode QUOTE_RATE_EXCEEDED = 4030;
        const exceptions::ErrorCode RATE_EXCEEDED = 4040;
        const exceptions::ErrorCode SEQUENCE_SIZE_EXCEEDED = 4050;
        const exceptions::ErrorCode QUOTE_BEING_PROCESSED = 4060;
        const exceptions::ErrorCode ORDER_BEING_PROCESSED = 4070;
        const exceptions::ErrorCode EXCHANGE_CLASS_GATE_CLOSED = 4080;
        const exceptions::ErrorCode SERVER_NOT_AVAILABLE = 4090;
        const exceptions::ErrorCode ACTION_VETOED = 4100;
        const exceptions::ErrorCode QUOTE_CONTROL_ID = 4110;
        const exceptions::ErrorCode UNSUPPORTED_INTERNALIZATION = 4120;
        const exceptions::ErrorCode AUCTION_INACTIVE = 4130;
        const exceptions::ErrorCode AUCTION_ENDED = 4140;

        //New ErrorCode for Single Acronym Scrum for Quote Entry Restriction.
        // If a class is being quoted by userId1 and any other userId sharing acronym tries to
        //send quote for same class, quote will be rejected with following error code.
    const exceptions::ErrorCode OTHER_USER_FOR_ACR_QUOTING_CLASS = 4150;
        const exceptions::ErrorCode EXCEEDS_CONCURRENT_QUOTE_LIMIT = 4160;
        const exceptions::ErrorCode QUOTE_CANCEL_IN_PROGRESS = 4170;

    // ErrorCodes for User Maintenance:
    //
    const exceptions::ErrorCode ONLY_USER_FOR_ACRONYM = 4200;
    const exceptions::ErrorCode USER_IS_ENABLED = 4210;
    const exceptions::ErrorCode USER_LOGGED_IN = 4220;
    const exceptions::ErrorCode USER_HAS_ORDER = 4230;
    const exceptions::ErrorCode RECENT_USER_ACTIVITY = 4240;

        // OHS PendingCancel ErrorCodes.
        const exceptions::ErrorCode PENDING_CANCEL = 4300;
        const exceptions::ErrorCode LOCATION_NOT_AVAILABLE = 4310;

        //Error code for Manul Quote Reporting
        //This code can not be changed as these codes are mapped with TPF contsants
        //for manul quote
        const exceptions::ErrorCode MANUAL_QUOTE_ACCEPTED = 6001;
        const exceptions::ErrorCode MANUAL_QUOTE_MARKETABLE = 6002;
        const exceptions::ErrorCode MANUAL_QUOTE_WORSE_THAN_MARKET = 6003;
        const exceptions::ErrorCode
MANUAL_QUOTE_MARKETABLE_WITH_STRATEGY = 6004;
        const exceptions::ErrorCode MANUAL_QUOTE_SYSTEM_ERROR = 6005;
        const exceptions::ErrorCode MANUAL_QUOTE_INVALID_REQUEST = 6006;
        const exceptions::ErrorCode MANUAL_QUOTE_NOT_ACCEPTED = 6007;
        const exceptions::ErrorCode MANUAL_QUOTE_OVERRIDE_NEEDED = 6008;
        const exceptions::ErrorCode
MANUAL_QUOTE_CLASS_NOT_IDX_HYBRID_ENABLED = 6009;

        // DirectedAIM NotAccepted Codes.
        const exceptions::ErrorCode DIRECTED_AIM_PRIMARY_EXPIRED = 7000;
        const exceptions::ErrorCode NOT_REGISTERED_FOR_DIRECTED_AIM =
7001;
        const exceptions::ErrorCode NO_PDPM_AVAILABLE_FOR_DIRECTED_AIM =
7002;
```

const exceptions::ErrorCode NO_DPM_AVAILABLE_FOR_DIRECTED_AIM = 7003;

const exceptions::ErrorCode NO_AFFILIATED_MM_AVAILABLE_FOR_DIRECTED_AIM = 7004;

const exceptions::ErrorCode USER_NOT_AFFILIATED_TO_ANY_FIRM = 7005;

const exceptions::ErrorCode INVALID_AFFILIATED_FIRM = 7006;

const exceptions::ErrorCode ALREADY_UPDATED_AS_REGISTERED = 7007;

const exceptions::ErrorCode ALREADY_UPDATED_AS_UNREGISTERED = 7008;

};

Document Changes

API-01

- No changes

API-02

- Modified the Blocked Cancel Request section to read "allow users to send in cancel requests for up to ~~40~~ 400 series per method call."
- Updated the AIM section to include Directed AIM features as described above.
- Added a new section for "Market Maker Hand Held Functionality" as described above.
- Created a new section for "CBOE 2 (C2)."
- Modified the section "Support for Multiple Trading Sessions" to include the new exchange session, C2_MAIN.
- Updated the section "Order Contingency Types Available in CBOE Trading Sessions" to include contingency types availabe for C2.
- Changed the SAL section for Non-Hybrid Index Classes to state 250 contracts instead of 50:
  - Eligible marketable orders will be stopped at the LMM quote and exposed to a brief (300ms) electronic aution for price improvement. Customer orders of **250** contracts and less will be eligible.

API-03

- Included definitions for the CMi V6 interfaces.
- Added values for the new constants based on this release.

API-04

- Included definitions for the CMi V6 interfaces.

- Added values for the new constants based on this release.

API-05

- No changes

API-06

- No changes

API-07

- No changes

API-08

- No changes

CAS-01

- No changes

CAS-02

- Changes to support the new interface, FloorTradeMaintenanceService.

Simulator

- A new simulator example (Example17) was created to demonstrate the CMi MMHH application for firms. The new example execution interface is consistent with the rest of the examples. Example17 demonstrates generating a MMHH trade, deleting a MMHH trade and subscribing/unsubscribing for MMHH trade notifications.

Test Plan Changes

- No changes