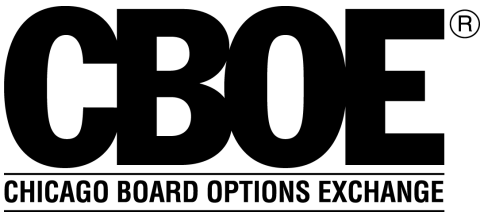
CONFIDENTIAL



CBOE Application Programming Interface

CBOE API Version 2.5 - Release Notes

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

CBOE PROPRIETARY INFORMATION

14 March 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

This document contains material excerpted from CBOEdirect Volume API02 as well as new insights into the upcoming release of the next version of the CMi api. The upcoming release will support the CBOE Hybrid trading environment as well as the existing ONE Chicago Futures market. The material is presented to highlight the upcoming changes and place emphasis on important design decisions. 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 interface.

CORBA Callback Interfaces

The CORBA Callback mechanism is used to implement a publish-subscribe mechanism between the CAS and your program. There are two common approaches to subscribing for information from the CAS.

The first is using an explicit subscribe method available via one of the CAS interfaces. For instance, if you want to subscribe for ticker data for a particular product, the following operation from the MarketQuery interface would be invoked:

```
MarketQuery::subscribeTicker(sessionName,productKey,clientListener);
```

Where:

`sessionName` is a `cmiSession::TradingSessionName` that specifies from which trading session the ticker data should be sent.

`productKey` is a `cmiProduct::ProductKey` that specifies the product whose ticker data you want to receive.

`clientListener` is the `Callback::CMITickerConsumer` object you have implemented within your program to process ticker information. The operation that the CAS will invoke on your `CMITickerConsumer` is:

```
acceptTicker(ticker)
```

Where:

`ticker` is a `cmiMarketData::TickerStructSequence` which is a sequence of `cmiMarketData::TickerStructs`

This pattern for subscription to information provided by the CAS is frequently repeated.

To subscribe to information from the CAS using a subscribe operation available through one of the CAS interfaces, you must:

- Implement the appropriate CORBA Callback object in your client application.
- Register that callback with the CAS by invoking the appropriate subscribe operation on the appropriate CAS interface.
- Implement the appropriate accept operation in the CORBA Callback object

The second pattern implemented for subscription by the CMi is the use of a query operation that returns a snapshot of information and simultaneously registers a client callback object to receive subsequent updates published by the CAS.

For instance, if you wanted to retrieve current product information for a product, you would invoke the following operation on the `ProductQuery` interface:

```
ProductQuery::getProductsForSession(sessionName, classKey,
includeActiveOnly, clientListener)
```

which returns a sequence of cmiProduct::ProductStructs

Where:

`sessionName` is a `cmiSession::TradingSessionName` that specifies from which trading session the list of products eligible for trading is to be retrieved.

`classKey` is a `cmiProduct::ClassKey` that indicates the class whose products (series) are to be returned.

`clientListener` is a `cmiCallback::CMIProductStatusConsumer` callback object that you implemented in your client program to accept updates to product status that are published by the CAS.

Which interfaces you implement will depend on what information to which you chose to subscribe or are required to subscribe.

Callback Performance Design Issues

Threading is an extremely important aspect of callbacks. Instances of callback objects, e.g. CMIOrderBookConsumer, are registered with the CMi to receive status and market data, e.g.

> MarketQuery.subscribeBookDepth(in cmiSession::TradingSessionName sessionName, in cmiProduct::ProductKey productKey, in cmiCallback::CMIOrderBookConsumer orderBookConsumer)

The CAS guarantees that for a given callback object, messages will be delivered in the order that they are received (by the CAS). To accomplish this, the CAS maintains a queue for each callback object registered, and delivers each message after the client has successfully processed the previous message. Each queue will be processed independently.

In theory, the client could use the same callback object for each product or class-based subscription. For example, the client application could instantiate a callback object, and subscribes for products for the classes IBM, AOL, and GE using this callback. If the callback in the client application is activated with a multithreaded POA, it has the ability to handle concurrent calls for all three classes. However, since the CAS sees this as a single callback, and associates the callback with a single queue, the calls would actually be serialized in the CAS. This is a particularly bad choice for high volumes of messages, especially as will be found in options trading.

The solution for this problem is to balance the load across multiple callback objects. Two different ways of accomplishing this might be to use a pool of callback objects or to have a distinct callback object per product class. It should be noted that each user session has a limited number of threads associated with it. If the CAS is trying to service too many callback objects it is possible for some high volume messages to become thread starved. CBOE's recommendation currently is to use a callback object per product class and message type. This would allow the CAS to invoke the callbacks on distinct threads, without the overhead of managing an excessive number of queues or the delay of one subject's messages behind another subject's messages.

Creating multiple callback objects in the client application does not necessarily mean creating a large number of servant objects. The CORBA object model clearly separates the concept of an object reference from that of a servant. Thus, multiple references could be created by the client, but associated with a single servant, or set of servants as long as the servants are thread safe and don't inadvertently synchronize threads against each other:

```
CMIOrderBookConsumer callbackServant = new CMIOrderBookConsumerImpl();
for ( int i = 1 ; i <= NUM_CLASSES ; i++ ) {
        byte[] servantId = ("Callback" + i).getBytes(); // create a new ID
                myPOA.activate_object_with_id(servantId, callbackServant);
        }
```

Given a high rate of messages, a user will need to design to allow for a configurable number of callback objects. The specific number will most likely be determined during your integration and load testing. The key points are:

- The CAS associates a thread with each unique callback.

- An excessive number of threads can affect performance due to the overhead of managing the threads
- Too few callbacks can affect performance due to the serialization of messages to a single callback.
- If your implementation associates multiple callback objects to a single servant instance, it is important that the servant be thread safe and not contain any code that could inadvertently synchronize the threads against each other.
- When you register a callback object you become a CORBA server. It is important to make sure that your Orb has been configured with enough threads in its POA threadpool to service all the incoming messages. One performance aspect often overlooked is the available CORBA ORB resources. Depending on your ORB the default setting can be limiting. Review your ORB documentation for limitations with regard to thread usage and adjust the settings to your use. While we do not provide a performance lab it is simple enough to create a simulator to drive your ORB (and your own application) to assist in the tuning.
- Another consideration is to limit the processing on each callback thread to the absolute minimum. The faster a callback returns the faster the CAS can send the next message for that subscription. The CAS message performance is directly related to how efficient the client listeners are.

Heartbeat Callback

The Orb on which the current implementation of the CAS is based did not have the ability to detect network failures or hung applications. As a result if the CMi client application were to hang or deadlock, the CAS would simply queue heartbeats to that application and never detect that it was no longer responding. Additionally if there was a loss of network connectivity, the orb had an infinite timeout. As a result the connection would hang until the socket physically closed at the OS level, which is usually set to be about 15 minutes.

The new Orb on which the CAS is now based uses a timeout for heartbeat. Currently this timeout is set to be 20 seconds. This means when a CMi client application doesn't respond to a single heartbeat request within 20 seconds for any reason, the connection will be considered dead and the CAS will logoff the connection and clean up its session. As a result, care must be taken to make sure that the heartbeat thread in your application never becomes starved or isn't serviced or your application may be logged off on a false detection.

Login IOR

The IOR is by standard a variable length message that can differ in implementation from ORB to ORB. The new release of the CAS and Simulator is built on a different ORB and returns a larger IOR string. This IOR will normally be split across multiple packets – use the supplied length field in the IOR string.

The IDL Interfaces

CMi Callbacks

As we have stated previously, we suggest using a unique callback per class wherever possible. Market data, Quote and Order Status messages are subscriptions where a class level object design still results in a fairly manageable number of threads. While this can result in 2000 plus threads in the options or equity world any other allocation will potentially result in the queuing of one classes messages behind another's. A case can be made for using fewer threads when dealing with low volume classes but that requires manual adjustments as the market changes.

One of the new features we are providing is a Queue Depth parameter on the callbacks. This value will inform you of the depth of any queue your CAS server currently has for your process. You will be able to set a threshold and an action to take to eliminate the queue.

CMi Constants IDL

```
QueueActions {
    cmiUtil::QueueAction NO_ACTION = 0;
    cmiUtil::QueueAction FLUSH_QUEUE = 1;
    cmiUtil::QueueAction OVERLAY_LAST = 2;
    cmiUtil::QueueAction DISCONNECT_CONSUMER = 3;
};



module cmiCallbackV2
{
    interface CMICurrentMarketConsumer {
        oneway void acceptCurrentMarket(
            in cmiMarketData::CurrentMarketStructSequence currentMarket,
            in long queueDepth,
            in cmiUtil::QueueAction queueAction);
    };

    interface CMINBBOConsumer {
        oneway void acceptNBBO(
            in cmiMarketData::NBBOStructSequence nbbo,
            in long queueDepth,
            in cmiUtil::QueueAction queueAction);
    };

    interface CMIExpectedOpeningPriceConsumer {
```

```
    oneway void acceptExpectedOpeningPrice(
        in cmiMarketData::ExpectedOpeningPriceStructSequence expectedOpeningPrices,
        in long queueDepth,
        in cmiUtil::QueueAction queueAction);
};

interface CMIRFQConsumer {
    oneway void acceptRFQ(
        in cmiQuote::RFQStructSequence rfqs,
        in long queueDepth,
        in cmiUtil::QueueAction queueAction);
};

interface CMIRecapConsumer {
    oneway void acceptRecap(
        in cmiMarketData::RecapStructSequence recap,
        in long queueLength,
        in cmiUtil::QueueAction queueAction);
};

interface CMITickerConsumer {
    oneway void acceptTicker(
        in cmiMarketData::TickerStructSequence ticker,
        in long queueDepth,
        in cmiUtil::QueueAction queueAction);
};

interface CMIOrderBookConsumer {
    oneway void acceptBookDepth(
        in cmiMarketData::BookDepthStructSequence productBooks,
        in long queueDepth,
        in cmiUtil::QueueAction queueAction);
};

interface CMIOrderBookUpdateConsumer {
    oneway void acceptBookDepthUpdate(
        in cmiMarketData::BookDepthUpdateStructSequence productBooks,
        in long queueDepth,
        in cmiUtil::QueueAction queueAction);
};

interface CMIOrderStatusConsumer {
```

```
        void acceptOrderStatus(
            in cmiOrder::OrderDetailStructSequence orders,
            in long queueDepth);

        void acceptOrderCanceledReport(
            in cmiOrder::OrderCancelReportStruct canceledReport,
            in long queueDepth);

        void acceptOrderFilledReport(
            in cmiOrder::OrderFilledReportStruct filledReport,
            in long queueDepth);

        void acceptOrderBustReport(
            in cmiOrder::OrderBustReportStruct bustReport,
            in long queueDepth);

        void acceptOrderBustReinstateReport(
            in cmiOrder::OrderBustReinstateReportStruct bustReinstatedReport,
            in long queueDepth);

        void acceptNewOrder(
            in cmiOrder::OrderDetailStruct order,
            in long queueDepth);
    };

    interface CMIQuoteStatusConsumer {
        // Quote Deletes will no longer be published as part of Quote Status
        // Listen for the acceptQuoteDeleteReport message instead
        // New Quote messages will only be published if 'includeBookedStatus'
        // is set to true on the subscribe call
        void acceptQuoteStatus( in cmiQuote::QuoteDetailStructSequence quotes,
            in long queueDepth);

        void acceptQuoteFilledReport(
            in cmiQuote::QuoteFilledReportStruct filledReport,
            in long queueDepth);

        void acceptQuoteBustReport(
            in cmiQuote::QuoteBustReportStruct bustReport,
            in long queueDepth);

        void acceptQuoteDeleteReport(
```

```
            in cmiQuote::QuoteDeleteReportStructSequence cancelReports,
            in long queueDepth);
    };

    interface CMILockedQuoteStatusConsumer {
        void acceptQuoteLockedReport(
            in cmiQuote::LockNotificationStructSequence lockedQuotes,
            in long queueDepth);
    };
};
```

Market Data

The market data interfaces previously allowed the user to register callback objects per class. However not all methods returned sequences of responses. Almost all methods have been modified to return sequences to allow us to take advantage of blocking where appropriate. For instance a single block of quotes from any user can result in multiple updates to the current market quote for several products within a class. We will deliver the market updates in a single message where possible.

The new parameters available in the market data subscriptions are:

Queue Action.

Queue Depth.

A new book depth data struct will be added to allow for more granular information. This information will be available on a request *not* on a subscription basis. Access to this call will be limited on a per minute basis and by user type.

Market Data IDL:

```
typedef short VolumeType;
typedef short CurrentMarketViewType;
typedef short OrderBookPriceViewType;

struct CurrentMarketViewStruct
{
    cmiMarketData::CurrentMarketViewType currentMarketViewType;
    cmiUtil::PriceStruct bidPrice;
    cmiMarketData::MarketVolumeStructSequence bidSizeSequence;
    cmiUtil::PriceStruct askPrice;
    cmiMarketData::MarketVolumeStructSequence askSizeSequence;
```

```
};
typedef  sequence <CurrentMarketViewStruct> CurrentMarketViewStructSequence;

struct CurrentMarketStructV2
{
    cmiProduct::ProductKeysStruct productKeys;
    cmiSession::TradingSessionName sessionName;
    string exchange;
    cmiMarketData::CurrentMarketViewStructSequence currentMarketViews;
    cmiUtil::TimeStruct sentTime;
    boolean bidIsMarketBest;
    boolean askIsMarketBest;
    boolean legalMarket;
};
typedef sequence <CurrentMarketStructV2> CurrentMarketStructV2Sequence;



struct OrderBookPriceViewStruct
{
    cmiMarketData::OrderBookPriceViewType orderBookPriceViewType;
    cmiMarketData::MarketVolumeStructSequence viewSequence;
};
typedef sequence <OrderBookPriceViewStruct> OrderBookPriceViewStructSequence;

struct OrderBookPriceStructV2
{
    cmiUtil::PriceStruct price;
    OrderBookPriceViewStructSequence views;
};
typedef sequence <OrderBookPriceStructV2> OrderBookPriceStructV2Sequence;

struct BookDepthStructV2
{
    cmiProduct::ProductKeysStruct productKeys;
    cmiSession::TradingSessionName sessionName;
    cmiMarketData::OrderBookPriceStructV2Sequence buySideSequence;
    cmiMarketData::OrderBookPriceStructV2Sequence sellSideSequence;
    boolean allPricesIncluded;
    long transactionSequenceNumber;
};

struct MarketVolumeStruct {
```

```
    cmiMarketData::VolumeType volumeType;
    long quantity;
    boolean multipleParties;
};
```

Market Data Service IDL

```
interface MarketQuery : cmi::MarketQuery
{
    void subscribeRecapForClassV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMIRecapConsumer clientListener,
        in cmiUtil::QueueAction actionOnQueue)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void unsubscribeRecapForClassV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMIRecapConsumer clientListener)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void subscribeRecapForProductV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ProductKey productKey,
        in cmiCallbackV2::CMIRecapConsumer clientListener,
        in cmiUtil::QueueAction actionOnQueue)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
```

```
            exceptions::DataValidationException
        );

    void unsubscribeRecapForProductV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ProductKey productKey,
        in cmiCallbackV2::CMIRecapConsumer clientListener)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void subscribeCurrentMarketForClassV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMICurrentMarketConsumer clientListener,
        in cmiUtil::QueueAction actionOnQueue)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void unsubscribeCurrentMarketForClassV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMICurrentMarketConsumer clientListener)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void subscribeCurrentMarketForProductV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ProductKey productKey,
        in cmiCallbackV2::CMICurrentMarketConsumer clientListener,
        in cmiUtil::QueueAction actionOnQueue)
```

```
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );

    void unsubscribeCurrentMarketForProductV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ProductKey productKey,
        in cmiCallbackV2::CMICurrentMarketConsumer clientListener)
           raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );

     void subscribeNBBOForClassV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMINBBOConsumer clientListener,
        in cmiUtil::QueueAction actionOnQueue)
            raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );

    void unsubscribeNBBOForClassV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMINBBOConsumer clientListener)
            raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );

    void subscribeNBBOForProductV2(
```

```
    in cmiSession::TradingSessionName sessionName,
    in cmiProduct::ProductKey productKey,
    in cmiCallbackV2::CMINBBOConsumer clientListener,
    in cmiUtil::QueueAction actionOnQueue)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );

 void unsubscribeNBBOForProductV2(
    in cmiSession::TradingSessionName sessionName,
    in cmiProduct::ProductKey productKey,
    in cmiCallbackV2::CMINBBOConsumer clientListener)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );

void subscribeTickerForProductV2(
    in cmiSession::TradingSessionName sessionName,
    in cmiProduct::ProductKey productKey,
    in cmiCallbackV2::CMITickerConsumer clientListener,
    in cmiUtil::QueueAction actionOnQueue)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );

void unsubscribeTickerForProductV2(
    in cmiSession::TradingSessionName sessionName,
    in cmiProduct::ProductKey productKey,
    in cmiCallbackV2::CMITickerConsumer clientListener)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
```

```
            exceptions::DataValidationException
        );

    void subscribeTickerForClassV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMITickerConsumer clientListener,
        in cmiUtil::QueueAction actionOnQueue)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void unsubscribeTickerForClassV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMITickerConsumer clientListener)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void subscribeExpectedOpeningPriceForProductV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ProductKey productKey,
        in cmiCallbackV2::CMIExpectedOpeningPriceConsumer clientListener,
        in cmiUtil::QueueAction actionOnQueue)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void unsubscribeExpectedOpeningPriceForProductV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ProductKey productKey,
        in cmiCallbackV2::CMIExpectedOpeningPriceConsumer clientListener)
```

```
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );

    void subscribeExpectedOpeningPriceForClassV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMIExpectedOpeningPriceConsumer clientListener,
        in cmiUtil::QueueAction actionOnQueue)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void unsubscribeExpectedOpeningPriceForClassV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMIExpectedOpeningPriceConsumer clientListener)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void subscribeBookDepthForClassV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMIOrderBookConsumer orderBookConsumer,
        in cmiUtil::QueueAction actionOnQueue)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
        );
```

```
void unsubscribeBookDepthForClassV2(
    in cmiSession::TradingSessionName sessionName,
    in cmiProduct::ClassKey classKey,
    in cmiCallbackV2::CMIOrderBookConsumer orderBookConsumer)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );

void subscribeBookDepthForProductV2(
    in cmiSession::TradingSessionName sessionName,
    in cmiProduct::ProductKey productKey,
    in cmiCallbackV2::CMIOrderBookConsumer orderBookConsumer,
    in cmiUtil::QueueAction actionOnQueue)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
    );

void unsubscribeBookDepthForProductV2(
    in cmiSession::TradingSessionName sessionName,
    in cmiProduct::ProductKey productKey,
    in cmiCallbackV2::CMIOrderBookConsumer orderBookConsumer)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );

void subscribeBookDepthUpdateForClassV2(
    in cmiSession::TradingSessionName sessionName,
    in cmiProduct::ClassKey classKey,
    in cmiCallbackV2::CMIOrderBookUpdateConsumer orderBookConsumer,
    in cmiUtil::QueueAction actionOnQueue)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
```

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

    void unsubscribeBookDepthUpdateForClassV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMIOrderBookUpdateConsumer orderBookConsumer)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void subscribeBookDepthUpdateForProductV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ProductKey productKey,
        in cmiCallbackV2::CMIOrderBookUpdateConsumer orderBookConsumer,
        in cmiUtil::QueueAction actionOnQueue)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void unsubscribeBookDepthUpdateForProductV2(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ProductKey productKey,
        in cmiCallbackV2::CMIOrderBookUpdateConsumer orderBookConsumer)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    cmiMarketData::BookDepthStructV2 getBookDepthDetails(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ProductKey productKey)
            raises(
```

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

};
```

Order Query

The order status subscription now allows multiple GMD callbacks to be registered. Instead of only allowing one GMD callback consumer per user, the CMi now allows the user to specify a separate GMD callback consumer per class. Additionally the decision on whether to publish existing orders on the subscription call to the consumer has been changed to a Boolean parameter.

Order Query Service IDL

```
interface OrderQuery : cmi::OrderQuery
{
   void subscribeOrderStatusV2(
       in cmiCallbackV2::CMIOrderStatusConsumer clientListener,
       in boolean publishOnSubscribe,
       in boolean gmdCallback)
           raises(
               exceptions::SystemException,
               exceptions::CommunicationException,
               exceptions::AuthorizationException,
               exceptions::DataValidationException
           );

   void unsubscribeOrderStatusV2(
       in cmiCallbackV2::CMIOrderStatusConsumer clientListener)
           raises(
               exceptions::SystemException,
               exceptions::CommunicationException,
               exceptions::AuthorizationException,
               exceptions::DataValidationException
           );

   void subscribeOrderStatusForClassV2(
```

```
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMIOrderStatusConsumer clientListener,
        in boolean publishOnSubscribe,
        in boolean gmdCallback)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void unsubscribeOrderStatusForClassV2(
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMIOrderStatusConsumer clientListener)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void subscribeOrderStatusForFirmV2(
        in cmiCallbackV2::CMIOrderStatusConsumer clientListener,
        in boolean publishOnSubscribe,
        in boolean gmdCallback)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void unsubscribeOrderStatusForFirmV2(
        in cmiCallbackV2::CMIOrderStatusConsumer clientListener)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void subscribeOrderStatusForFirmForClassV2(
```

```
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMIOrderStatusConsumer clientListener,
        in boolean publishOnSubscribe,
        in boolean gmdCallback)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );

    void unsubscribeOrderStatusForFirmForClassV2 (
        in cmiProduct::ClassKey classKey,
        in cmiCallbackV2::CMIOrderStatusConsumer clientListener)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
            );
};
```

Quotes

The quote status subscription now allows multiple GMD callbacks to be registered. Instead of only allowing one GMD callback consumer per user, the CMi now allows the user to specify a separate GMD callback consumer per class. Additionally, the indication that a quote has been cancelled or deleted will no longer be communicated through the quote status message. Instead it will be published to the acceptQuoteDeleteReport method on the new quote status callback consumer.

A new quote callback interface was also added to notify market makers when their quotes were locked against other quotes. This is on a separate interface to allow the user to service this callback independent of the fill messages.

The decision on whether to publish existing orders on the subscription call to the consumer has been changed to a Boolean parameter. Finally, a new parameter “includeBookedStatus” was added to allow the user to indicate to the CAS whether or not to publish the “booked” message through the quote status consumer. A successful call to acceptQuote with no exceptions (or a successful call to acceptQuotesForClass with no exceptions) with ErrorCodes all equal to 0 indicates the quote message has been processed and booked by the server. The booked status message was designed to be used by position management

systems, which usually are separate from the quoting application. Set the "includeBookedStatus" to true if you need the message.

Quote IDL

```
struct ClassQuoteResultStructV2
    {
        cmiQuote::QuoteKey quoteKey;
        cmiProduct::ProductKey productKey;
        exceptions::ErrorCode errorCode;
    };
    typedef sequence <ClassQuoteResultStructV2> ClassQuoteResultStructV2Sequence;


  Struct QuoteDeleteReportStruct {
      cmiQuote::QuoteDetailStruct quote;
      cmiUtil::ActivityReason deleteReason;
  };
  typedef sequence <QuoteDeleteReportStruct> QuoteDeleteReportStructSequence;
  struct LockNotificationStruct {
      cmiSession::TradingSessionName sessionName;
      cmiProduct::ProductType productType;
      cmiProduct::ClassKey classKey;
      cmiProduct::ProductKey productKey;
      cmiUtil::Side side;
      cmiUtil::PriceStruct price;
      long quantity;
      string extensions;
      cmiUtil:StringSequence:: buySideUserAcronyms;
      cmiUtil:StringSequence:: sellSideUserAcronyms;
  };
  typedef sequence <LockNotificationStruct> LockNotificationStructSequence;
```

Quote Query Service IDL

```
interface Quote : cmi::Quote {

cmiQuote::ClassQuoteResultStructV2Sequence acceptQuotesForClassV2(
        in cmiProduct::ClassKey classKey,
        in cmiQuote::QuoteEntryStructSequence quotes )
            raises(
```

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

    void subscribeQuoteStatusV2 (
        in cmiCallbackV2::CMIQuoteStatusConsumer clientListener,
        in boolean publishOnSubscribe,
        in boolean includeBookedStatus,
        in boolean gmdCallback)
            raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::DataValidationException,
            exceptions::AuthorizationException
        );

    void unsubscribeQuoteStatusV2 (
        in cmiCallbackV2::CMIQuoteStatusConsumer clientListener)
            raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );
    void subscribeQuoteStatusForFirmV2 (
            in cmiCallbackV2::CMIQuoteStatusConsumer clientListener,
            in boolean gmdCallback)
                raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::DataValidationException,
                exceptions::AuthorizationException
                );

    void unsubscribeQuoteStatusForFirmV2 (
            in cmiCallbackV2::CMIQuoteStatusConsumer clientListener)
                raises(
                exceptions::SystemException,
```

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


void subscribeQuoteStatusForClassV2 (
    in cmiProduct::ClassKey classKey,
    in boolean publishOnSubscribe,
    in boolean includeBookedStatus,
    in cmiCallbackV2::CMIQuoteStatusConsumer clientListener,
    in boolean gmdCallback)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::DataValidationException,
            exceptions::AuthorizationException
        );

void unsubscribeQuoteStatusForClassV2 (
    in cmiProduct::ClassKey classKey,
    in cmiCallbackV2::CMIQuoteStatusConsumer clientListener)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );

void subscribeQuoteStatusForFirmForClassV2 (
    in cmiProduct::ClassKey classKey,
    in cmiCallbackV2::CMIQuoteStatusConsumer clientListener,
    in boolean publishOnSubscribe,
    in boolean includeBookedStatus,
    in boolean gmdCallback)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::DataValidationException,
            exceptions::AuthorizationException
        );
```

```
void unsubscribeQuoteStatusForFirmForClassV2 (
    in cmiProduct::ClassKey classKey,
    in cmiCallbackV2::CMIQuoteStatusConsumer clientListener)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );
void subscribeQuoteLockedNotification (
            in boolean publishOnSubscribe,
            in cmiCallbackV2::CMILockedQuoteStatusConsumer clientListener,
            in boolean gmdCallback)
                raises(
                    exceptions::SystemException,
                    exceptions::CommunicationException,
                    exceptions::DataValidationException,
                    exceptions::AuthorizationException
                );


void unsubscribeQuoteLockedNotification (
            in cmiCallbackV2::CMILockedQuoteStatusConsumer clientListener,
                raises(
                    exceptions::SystemException,
                    exceptions::CommunicationException,
                    exceptions::AuthorizationException
                    exceptions::DataValidationException,
                  );


void subscribeQuoteLockedNotificationForClass (
    in cmiProduct::ClassKey classKey,
    in boolean publishOnSubscribe,
    in cmiCallbackV2::CMILockedQuoteStatusConsumer clientListener,
    in boolean gmdCallback)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::DataValidationException,
            exceptions::AuthorizationException
        );
```

```
void unsubscribeQuoteLockedNotificationForClass (
    in cmiProduct::ClassKey classKey,
    in cmiCallbackV2::CMILockedQuoteStatusConsumer clientListener)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
        );

        void subscribeRFQV2(
            in cmiSession::TradingSessionName sessionName,
            in cmiProduct::ClassKey classKey,
            in cmiCallbackV2::CMIRFQConsumer clientListener )
                raises(
                    exceptions::SystemException,
                    exceptions::CommunicationException,
                    exceptions::AuthorizationException,
                    exceptions::DataValidationException
                );

        void unsubscribeRFQV2(
            in cmiSession::TradingSessionName sessionName,
            in cmiProduct::ClassKey classKey,
            in cmiCallbackV2::CMIRFQConsumer clientListener)
                raises(
                    exceptions::SystemException,
                    exceptions::CommunicationException,
                    exceptions::AuthorizationException,
                    exceptions::DataValidationException
                );    };
```

Session Management

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 enhanced interfaces by getting a reference to the UserSessionManagerV2. This reference can be obtained in one of two ways. Both options are exposed on the UserAccessV2 object which will be made available as an alternate IOR link on the HTTP port that the CAS is publishing on.

The first option of accessing the new interfaces is to log into the old CMi and obtain a UserSessionManager reference. Using this reference, the CMi user can call

getUserSessionManagerV2, which will return a SessionManagerStructV2. This struct will contain the reference to the original CMi's UserSessionManager as well as the new enhanced UserSessionMangerV2. The other option is to logon using the UserAccessV2 interface. The new logon method takes exactly the same parameters as the old CMi's logon method and returns a newly enhanced SessionManagerStructV2 that can then be used to access both the old and new CMi methods.

Session Management IDL

```
interface UserSessionManagerV2
{
    cmiV2::Quote getQuoteV2()
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException
        );

    cmiV2::OrderQuery getOrderQueryV2()
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException
        );

    cmiV2::MarketQuery getMarketQueryV2()
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException
        );
};

struct SessionManagerStructV2
{
    cmi::UserSessionManager sessionManager;
    cmiV2::UserSessionManagerV2 sessionManagerV2;
};

interface UserAccessV2
{
    SessionManagerStructV2 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
                );

        UserSessionManagerV2 getUserSessionManagerV2(
            in cmi::UserSessionManager session )
                raises(
                    exceptions::SystemException,
                    exceptions::CommunicationException,
                    exceptions::AuthorizationException,
                    exceptions::NotFoundException
                );

    };
```

Test Plans

CMi Phase 2 Test Plans

- No changes

3a, Security Definition Test Plan

- FIX 4.2 firms cannot receive Pending Price Adjustments at this time.

3b, Market Data Test Plan

- Added strategy book depth
- If a firm wants to subscribe for large amounts of market data (for example: all products in all classes in a particular), the firm should sign up with CBOE Financial Network (CFN) or CBOE Ticker Express (details coming soon).

3c, Quote Test Plan (ETH / ONE)

- A firm's quoting application must use a Logon Session Mode "Primary", because if the application fails, all of the User's quotes will be automatically canceled by CBOE.
- Added a step for a user to enter a quote during product state of "Pre-Open".

3d, RTH Order Test Plan

- All testing steps were changed to make the document much shorter.

3e, ETH / ONE Order Test Plan

- The firm's order entry application must be Logon Session Mode "Primary" so the application can receive Guaranteed Message Delivery (GMD) messages upon logon.
- Added a step whereby a user enters orders with different executing give up exchanges and firms.
- Nothing Done tests should only be performed when the test environment is recycled, and should be skipped for phase 3x acceptance testing.

3f, Clearing Firm, Duplicate Message Test Plan

- No changes

3g, Strategy Quote Test Plan (ETH / ONE)

- Two steps were added (for futures and options) that test the firm's ability to enter a strategy quote with a same price of 0.00 and another step entering an opposite price of 0.00.
- If a CMi firm wishes to subscribe to receive new strategy products that are created by any user and also wishes to subscribe to receive strategy product states, it would need to make 2 separate message calls. The getStrategiesByClassForSession only subscribes the user for new strategy products for that class that are created intra-day (not product states), while the other call, getProductsForSession, subscribes the user for product states only (no new strategy products).

3h, RTH Strategy Order Test Plan

- No changes

3i, ETH / ONE Strategy Order Test Plan

- Nothing Done tests should only be performed when the test environment is recycled, and should be skipped for phase 3x acceptance testing.
- If a CMi firm wishes to subscribe to receive new strategy products that are created by any user and also wishes to subscribe to receive strategy product states, it would need to make 2 separate message calls. The getStrategiesByClassForSession only subscribes the user for new strategy products for that class that are created intra-day (not product states), while the other call, getProductsForSession, subscribes the user for product states only (no new strategy products).

4a, Assurance Production Application Test Plan

- This document was merged with Failure / Recovery and Quote Limit Test Plan to create the Phase 4 Test Plan document.

Phase 4b & 4c, Failure / Recovery and Quote Limit Test Plan

- This document was merged with Assurance Production Application Test Plan to create the Phase 4 Test Plan document.

Phase 4 Test Plan

- This is a new test plan document that was created by merging the Assurance Production Application Test Plan and the Failure / Recovery and Quote Limit Test Plan.
- Book Depth section was removed
- Sending RFQs section was removed
- Order Parameters section was removed
- Quote Limit changed to 50 quotes per second per user