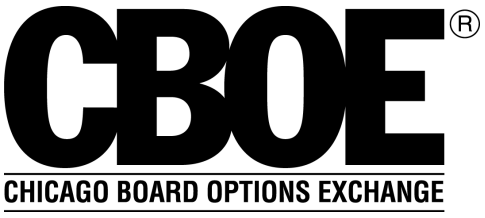
CONFIDENTIAL



CBOE Application Programming Interface

CBOE API Version 3.1 - Release Notes

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

CBOE PROPRIETARY INFORMATION

17 December 2004

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 changes for the new release of the CMi API, Version 3.1. This release supports functional upgrades for the Hybrid Trading environment. IDL, documentation and simulator changes for the CMi V3.1 are detailed in the sections below. Your feedback or questions regarding this document should be sent to api@cboe.com.

Firms wishing to connect to the current CMi API production system should use the Version 2.52 documents and IDL. The documentation and IDL is available for download on the API web site at http://systems.cboe.com/webAPI/.

Below are descriptions of the current CMi API releases that are available for download on the API website at http://systems.cboe.com/webAPI/.

- V2.5 – Current production version, including simulator.
- V2.52 – Current production upgrade of constants and error codes, no simulator
- V2.62 – Stock simulator
- V3.1 – Hybrid and Stock updates with simulator – ***this release***

CMi V3.1 Highlights

This version of the CMi details IDL changes for the internalization and auction process using CBOEdirect.

Order Entry

Internalization Order Entry

Internalization order entry will be accomplished via the new acceptInternalizationOrder method on interface com.cboe.idl.cmiV3.OrderEntry. The acceptInternalizationOrder method call must contain a primary (customer) order and a match (firm) order. The user submitting the two orders wishes to trade the match order with the primary order.

The acceptInternalizationOrder method call must contain a MatchType. Currently, LIMIT_PRICE and AUTO_MATCH are the only two MatchTypes supported. If the MatchType is LIMIT_PRICE, the match order price must be a limit price. If the MatchType is AUTO_MATCH, the match order price must be a market price. In both cases, the primary order price may be a limit price or a market price.

The acceptInternalizationOrder method returns an InternalizationOrderResultStruct which contains two OrderResultStructs, one for the primary order and one for the match order. Each OrderResultStruct will contain a valid OrderIdStruct for the order with which it is associated. Additionally, OrderResultStructs for valid orders will contain errorCode values of 0 and empty strings for their errorMessage fields

CMi V3

```
module cmiV3
interface OrderEntry : cmi::OrderEntry
  {
    cmiOrder::InternalizationOrderResultStruct
acceptInternalizationOrder(
        in cmiOrder::OrderEntryStruct primaryOrder,
        in cmiOrder::OrderEntryStruct matchOrder,
        in cmiOrder::MatchType matchType)
          raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException,
            exceptions::NotAcceptedException,
            exceptions::TransactionFailedException
          );
  };
// For internalization Orders call return
  struct OrderResultStruct
  {
    cmiOrder::OrderIdStruct orderId;
    cmiUtil::OperationResultStruct result;
  };
    typedef sequence <OrderResultStruct> OrderResultStructSequence;

  struct InternalizationOrderResultStruct
  {
    cmiOrder::OrderResultStruct primaryOrderResult;
    cmiOrder::OrderResultStruct matchOrderResult;
  };
```

Order Query

Auction Events

Users may subscribe for Request For Price (Auction) events by trading session and class. Any combination of supported auction types may be specified in the subscription. If the wrong session or class is submitted, a DataValidationException will be thrown and the subscription request will not be processed. Likewise, if any other exception is thrown during subscription, the subscription will have failed for all auction types. If no failures occur, but any auction type is invalid, the method will return. The result of the subscription for each auction type will be indicated in the corresponding OperationResultStruct for that auction type in the AuctionSubscriptionResultStructSequence. Each auction type the user submitted in the subscription call will be represented in this sequence, and any validation failure will be indicated by a nonzero error code and an error message in the OperationResultStruct. Subscriptions for any auction type that are successful will be indicated by an error code of 0. The user's callback will be subscribed for all auction types that were not marked invalid in the AuctionSubscriptionResultStructSequence.

CMi V3

```
module cmiV3
interface OrderQuery : cmiV2::OrderQuery
  {
   cmiOrder::AuctionSubscriptionResultStructSequence subscribeAuctionForClass(
      in cmiSession::TradingSessionName sessionName,
      in cmiProduct::ClassKey classKey,
      in cmiOrder::AuctionTypeSequence auctionTypes,
      in cmiCallbackV3::CMIAuctionConsumer clientListener)
         raises(
            exceptions::SystemException,
            exceptions::DataValidationException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException
         );

    cmiOrder::AuctionSubscriptionResultStructSequence unsubscribeAuctionForClass (
         in cmiSession::TradingSessionName sessionName,
         in cmiProduct::ClassKey classKey,
         in cmiOrder::AuctionTypeSequence auctionTypes,
         in cmiCallbackV3::CMIAuctionConsumer clientListener)
         raises(
            exceptions::SystemException,
            exceptions::DataValidationException,
```

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

Auction Response

Users who wish to participate in an auction may respond to Auction events by calling acceptOrder on the interface com.cboe.idl.cmiV3.OrderEntry. The OrderEntryStruct must be populated as if it were a normal order with the following modifications.

The OrderEntryStruct must contain an OrderContingencyStruct with type=ContingencyTypes.AUCTION_RESPONSE. The side of the auction response order must be tradable against the side indicated in the auction event.

The OrderEntryStruct extensions field must contain a new field for auction response orders corresponding to the auction ID of the auction. The extensions field should contain the substring "auctionId=123:456" where 123 is the high CBOE Id and 456 is the low CBOE Id of the auctionID specified in the auction event. The standard field separator ("\u0001") must be used in between subfields of the extensions field.

IDL Interfaces

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

cmiV3.idl

```
module cmiV3
{
   interface Quote : cmiV2::Quote
   {
      cmiQuote::ClassQuoteResultStructV3Sequence acceptQuotesForClassV3(
         in cmiProduct::ClassKey classKey,
         in cmiQuote::QuoteEntryStructV3Sequence quotes )
            raises(
               exceptions::SystemException,
               exceptions::CommunicationException,
               exceptions::AuthorizationException,
               exceptions::DataValidationException,
               exceptions::NotAcceptedException,
               exceptions::TransactionFailedException
            );

      void cancelAllQuotesV3(
```

```
        in cmiSession::TradingSessionName sessionName)
          raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException,
            exceptions::NotAcceptedException,
            exceptions::TransactionFailedException
          );
};
```

interface OrderQuery : cmiV2::OrderQuery

{

cmiOrder::AuctionSubscriptionResultStructSequence subscribeAuctionForClass(

in cmiSession::TradingSessionName sessionName,
in cmiProduct::ClassKey classKey,

in cmiOrder::AuctionTypeSequence auctionTypes,

in cmiCallbackV3::CMIAuctionConsumer clientListener)

raises(

exceptions::SystemException,

exceptions::DataValidationException,

exceptions::CommunicationException,

exceptions::AuthorizationException

);

This method is used to subscribe for auction solicitation from CBOE. The method will take a sequence of auction type. A data validation exception will be thrown if the wrong session or class is submitted. A result struct will be generated if the auction type is invalid.

cmiOrder::AuctionSubscriptionResultStructSequence unsubscribeAuctionForClass (

in cmiSession::TradingSessionName sessionName,

in cmiProduct::ClassKey classKey,

in cmiOrder::AuctionTypeSequence auctionTypes,

in cmiCallbackV3::CMIAuctionConsumer clientListener)

raises(

exceptions::SystemException,

exceptions::DataValidationException,

exceptions::CommunicationException,

```
            exceptions::AuthorizationException
        );
```

This method is used to unsubscribe from receiving auction solicitation from CBOE. The method will take a sequence of auction types. A data validation exception will be thrown if the wrong session or class is submitted. A result struct will be generated if the auction type is invalid.

```
};

interface OrderEntry : cmi::OrderEntry
{
   cmiOrder::InternalizationOrderResultStruct acceptInternalizationOrder(
      in cmiOrder::OrderEntryStruct primaryOrder,
      in cmiOrder::OrderEntryStruct matchOrder,
      in cmiOrder::MatchType matchType)
         raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException,
            exceptions::NotAcceptedException,
            exceptions::TransactionFailedException
         );
};
```

This method is used to submit internalization orders.

```
interface MarketQuery : cmiV2::MarketQuery
{
   cmiMarketData::MarketDataHistoryDetailStruct getDetailMarketDataHistoryByTime(
            in cmiSession::TradingSessionName sessionName,
            in cmiProduct::ProductKey productKey,
            in cmiUtil::DateTimeStruct startTime,
            in cmiUtil::QueryDirection direction)
            raises(
               exceptions::SystemException,
               exceptions::CommunicationException,
```

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

    cmiMarketData::MarketDataHistoryDetailStruct getPriorityMarketDataHistoryByTime(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ProductKey productKey,
        in cmiUtil::DateTimeStruct startTime,
        in cmiUtil::QueryDirection direction)
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::DataValidationException,
            exceptions::NotFoundException,
            exceptions::AuthorizationException
        );

void subscribeCurrentMarketForClassV3(
  in cmiSession::TradingSessionName sessionName,
  in cmiProduct::ClassKey classKey,
  in cmiCallbackV3::CMICurrentMarketConsumer clientListener,
  in cmiUtil::QueueAction actionOnQueue)
    raises(
      exceptions::SystemException,
      exceptions::CommunicationException,
      exceptions::AuthorizationException,
      exceptions::DataValidationException
    );

void unsubscribeCurrentMarketForClassV3(
  in cmiSession::TradingSessionName sessionName,
  in cmiProduct::ClassKey classKey,
  in cmiCallbackV3::CMICurrentMarketConsumer clientListener)
    raises(
      exceptions::SystemException,
```

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

    void subscribeCurrentMarketForProductV3(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ProductKey productKey,
        in cmiCallbackV3::CMICurrentMarketConsumer clientListener,
        in cmiUtil::QueueAction actionOnQueue)
          raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
          );

    void unsubscribeCurrentMarketForProductV3(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ProductKey productKey,
        in cmiCallbackV3::CMICurrentMarketConsumer clientListener)
          raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException
          );

};
interface UserSessionManagerV3 : cmiV2::UserSessionManagerV2, cmi::UserSessionManager
{
    cmiV3::MarketQuery getMarketQueryV3()
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException
```

```
        );

    cmiV3::Quote getQuoteV3()
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException
        );

    cmiV3::OrderQuery getOrderQueryV3()
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException
        );

    cmiV3::OrderEntry getOrderEntryV3()
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException
        );
};

interface UserAccessV3
{
    UserSessionManagerV3 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
            );

    };
```

cmiCallbackV3.idl

```
module cmiCallbackV3
{
  interface CMICurrentMarketConsumer {
    void acceptCurrentMarket(
      in cmiMarketData::CurrentMarketStructSequence bestMarkets,
      in cmiMarketData::CurrentMarketStructSequence bestPublicMarkets,
      in long queueDepth,
      in cmiUtil::QueueAction queueAction);
  };
  interface CMIAuctionConsumer
  {
    void acceptAuction(in cmiOrder::AuctionStruct auctionRequest);
  };
```

cmiOrder.idl

```
module cmiOrder
{
  typedef short ContingencyType;
  typedef short OrderState;
  typedef char  TimeInForce;
  typedef char  PositionEffect;
  typedef char  OriginType;
  typedef char  Coverage;
  typedef boolean  CrossingIndicator;
  typedef short CancelType;
  typedef short NBBOProtectionType;
  typedef short AuctionType;
  typedef short AuctionState;
```

```
typedef sequence <cmiOrder::OriginType> OriginTypeSequence;
typedef sequence <cmiOrder::AuctionType> AuctionTypeSequence;


struct OrderContingencyStruct
{
   cmiOrder::ContingencyType type;
   cmiUtil::PriceStruct price;
   long volume;
};

struct OrderIdStruct
{
   cmiUser::ExchangeFirmStruct executingOrGiveUpFirm;
   string branch;
   long branchSequenceNumber;
   string correspondentFirm;
   string orderDate; // YYYYMMDD format
   long highCboeId;
   long lowCboeId;
};

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;

struct LegOrderEntryStruct
{
    cmiProduct::ProductKey productKey;
    cmiUtil::PriceStruct mustUsePrice;
    cmiUser::ExchangeFirmStruct clearingFirm;
    cmiOrder::Coverage coverage;
    cmiOrder::PositionEffect positionEffect;
};
typedef sequence <LegOrderEntryStruct> LegOrderEntryStructSequence;

struct LegOrderDetailStruct
{
    cmiProduct::ProductKey productKey;
    cmiUtil::PriceStruct mustUsePrice;
    cmiUser::ExchangeFirmStruct clearingFirm;
    cmiOrder::Coverage coverage;
    cmiOrder::PositionEffect positionEffect;
    cmiUtil::Side side;
    long originalQuantity;
    long tradedQuantity;
```

```
    long cancelledQuantity;
    long leavesQuantity;
};
typedef sequence <LegOrderDetailStruct> LegOrderDetailStructSequence;

struct OrderStruct
{
    OrderIdStruct orderId;
    cmiUser::ExchangeAcronymStruct originator;

    // Fields from the OrderEntryStruct
    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;

    // Additional Order Fields
    string userId;
    cmiUser::ExchangeAcronymStruct userAcronym;
    cmiProduct::ProductType productType;
    cmiProduct::ClassKey classKey;
    cmiUtil::DateTimeStruct receivedTime;
    cmiOrder::OrderState state;
```

```
    long tradedQuantity;
    long cancelledQuantity;
    long leavesQuantity;
    cmiUtil::PriceStruct averagePrice;
    long sessionTradedQuantity;
    long sessionCancelledQuantity;
    cmiUtil::PriceStruct sessionAveragePrice;
    string orsId;
    cmiUtil::Source source;
    cmiOrder::OrderIdStruct crossedOrder;
    long transactionSequenceNumber;
    string userAssignedId;
    cmiSession::TradingSessionNameSequence sessionNames;
    cmiSession::TradingSessionName activeSession;
    cmiOrder::LegOrderDetailStructSequence legOrderDetails;
};
typedef sequence <OrderStruct> OrderStructSequence;

struct OrderDetailStruct
{
    cmiProduct::ProductNameStruct productInformation;
    cmiUtil::UpdateStatusReason statusChange;
    cmiOrder::OrderStruct orderStruct;
};
typedef sequence <OrderDetailStruct> OrderDetailStructSequence;

struct CancelReportStruct
{
    cmiOrder::OrderIdStruct orderId;
    cmiUtil::ReportType cancelReportType;
    cmiUtil::ActivityReason cancelReason;
    cmiProduct::ProductKey productKey;
    cmiSession::TradingSessionName sessionName;
    long cancelledQuantity;
    long tlcQuantity;
    long mismatchedQuantity;
```

```
    cmiUtil::DateTimeStruct timeSent;
    string orsId;
    long totalCancelledQuantity;
    long transactionSequenceNumber;
    string userAssignedCancelId;
};
typedef sequence <CancelReportStruct> CancelReportStructSequence;

struct CancelRequestStruct
{
    cmiOrder::OrderIdStruct orderId;
    cmiSession::TradingSessionName sessionName;
    string userAssignedCancelId;
    cmiOrder::CancelType cancelType;
    long quantity;
};

struct ContraPartyStruct
{
    cmiUser::ExchangeAcronymStruct user;
    cmiUser::ExchangeFirmStruct firm;
    long quantity;
};
typedef sequence <ContraPartyStruct> ContraPartyStructSequence;

struct FilledReportStruct
{
    cmiUtil::CboeIdStruct tradeId;
    cmiUtil::ReportType fillReportType;
    cmiUser::ExchangeFirmStruct executingOrGiveUpFirm;
    string userId;
    cmiUser::ExchangeAcronymStruct  userAcronym;
    cmiProduct::ProductKey productKey;
    cmiSession::TradingSessionName sessionName;
    long tradedQuantity;
    long leavesQuantity;
```

```
    cmiUtil::PriceStruct price;
    cmiUtil::Side side;
    string orsId;
    string executingBroker;
    cmiUser::ExchangeFirmStruct cmta;
    string account;
    string subaccount;
    cmiUser::ExchangeAcronymStruct originator;
    string optionalData;
    string userAssignedId;
    string extensions;
    cmiOrder::ContraPartyStructSequence contraParties;
    cmiUtil::DateTimeStruct timeSent;
    cmiOrder::PositionEffect positionEffect;
    long transactionSequenceNumber;
};
typedef sequence <FilledReportStruct> FilledReportStructSequence;

struct OrderFilledReportStruct
{
    cmiOrder::OrderDetailStruct filledOrder;
    cmiOrder::FilledReportStructSequence filledReport;
};
typedef sequence <OrderFilledReportStruct> OrderFilledReportStructSequence;

struct OrderCancelReportStruct
{
    cmiOrder::OrderDetailStruct cancelledOrder;
    cmiOrder::CancelReportStructSequence cancelReport;
};
typedef sequence <OrderCancelReportStruct> OrderCancelReportStructSequence;

struct PendingOrderStruct {
    cmiProduct::PendingNameStruct pendingProductName;
    cmiOrder::OrderStruct pendingOrder;
    cmiOrder::OrderStruct currentOrder;
```

```
};
typedef sequence <PendingOrderStruct> PendingOrderStructSequence;

struct BustReportStruct
{
   cmiUtil::CboeIdStruct tradeId;
   cmiUtil::ReportType bustReportType;
   cmiSession::TradingSessionName sessionName;
   cmiUser::ExchangeFirmStruct executingOrGiveUpFirm;
   string userId;
   cmiUser::ExchangeAcronymStruct  userAcronym;
   long bustedQuantity;
   cmiUtil::PriceStruct price;
   cmiProduct::ProductKey productKey;
   cmiUtil::Side side;
   cmiUtil::DateTimeStruct timeSent;
   long reinstateRequestedQuantity;
   long transactionSequenceNumber;
};
typedef sequence <BustReportStruct> BustReportStructSequence;

struct OrderBustReportStruct
{
   cmiOrder::OrderDetailStruct bustedOrder;
   cmiOrder::BustReportStructSequence bustedReport;
};
typedef sequence <OrderBustReportStruct> OrderBustReportStructSequence;

struct BustReinstateReportStruct
{
   cmiUtil::CboeIdStruct tradeId;
   long bustedQuantity;
   long reinstatedQuantity;
   long totalRemainingQuantity;
   cmiUtil::PriceStruct price;
   cmiProduct::ProductKey productKey;
```

```
    cmiSession::TradingSessionName sessionName;
    cmiUtil::Side side;
    cmiUtil::DateTimeStruct timeSent;
    long transactionSequenceNumber;
};

typedef sequence <BustReinstateReportStruct> BustReinstateReportStructSequence;

struct OrderBustReinstateReportStruct
{
    cmiOrder::OrderDetailStruct reinstatedOrder;
    cmiOrder::BustReinstateReportStruct bustReinstatedReport;
};

typedef sequence <OrderBustReinstateReportStruct>
OrderBustReinstateReportStructSequence;

typedef short MatchType; // can be auto-match, fixed limit price match, or guaranteed
auction starting price match

struct AuctionStruct
{
    cmiSession::TradingSessionName sessionName;
    cmiProduct::ClassKey classKey;
    cmiProduct::ProductType productType;
    cmiProduct::ProductKey productKey;
    cmiUtil::CboeIdStruct auctionId;
    cmiOrder::AuctionType auctionType;
    cmiOrder::AuctionState auctionState;
    cmiUtil::Side side;
    long auctionQuantity;
    cmiUtil::PriceStruct startingPrice;
    cmiOrder::ContingencyType auctionedOrderContingencyType;
    cmiUtil::TimeStruct entryTime;
    string extensions;
};
typedef sequence <AuctionStruct> AuctionStructSequence;
```

```
    // For internalization Orders call return
    struct OrderResultStruct
    {
       cmiOrder::OrderIdStruct orderId;
      cmiUtil::OperationResultStruct result;
    };
    typedef sequence <OrderResultStruct> OrderResultStructSequence;


    struct InternalizationOrderResultStruct
    {
       cmiOrder::OrderResultStruct primaryOrderResult;
       cmiOrder::OrderResultStruct matchOrderResult;
    };
typedef sequence <InternalizationOrderResultStruct>
InternalizationOrderResultStructSequence;


  struct AuctionSubscriptionResultStruct
  {
     cmiOrder::AuctionType auctionType;
     cmiUtil::OperationResultStruct subscriptionResult;
  };
  typedef sequence <AuctionSubscriptionResultStruct>
AuctionSubscriptionResultStructSequence;


 };
```

cmiConstants.idl

```
 interface ContingencyTypes
    {
       const cmiOrder::ContingencyType NONE = 1; // no contingency
       const cmiOrder::ContingencyType AON = 2; // All or None
       const cmiOrder::ContingencyType FOK = 3; // Fill or Kill
       const cmiOrder::ContingencyType IOC = 4; // Immediate or Cancel
       const cmiOrder::ContingencyType OPG = 5; // Opening only
       const cmiOrder::ContingencyType MIN = 6; // Minimum
```

```
        const cmiOrder::ContingencyType NOTHELD = 7; // Not held
        const cmiOrder::ContingencyType WD = 8; // With discretion
        const cmiOrder::ContingencyType MIT = 9; // Market if touched
        const cmiOrder::ContingencyType STP = 10; // Stop order
        const cmiOrder::ContingencyType STP_LOSS = 11; // Stop loss
        const cmiOrder::ContingencyType CLOSE = 12; // On close
        const cmiOrder::ContingencyType STP_LIMIT = 13; // Stop limit
        const cmiOrder::ContingencyType AUCTION_RESPONSE = 14; // Auction response
order


    };

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";

        // 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 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";
    const ExtensionField AUCTION_ID = "auctionId"; // used for auction response

};

interface MatchTypes
  {
    // the default match type for internalization
    const cmiOrder::MatchType UNSPECIFIED = 0;     /* not used */
```

This constant is used as the default match type for internatilization. It is not currently used.

```
    const cmiOrder::MatchType GUARANTEE_STARTING_PRICE = 1;   /* not currently used */
    const cmiOrder::MatchType LIMIT_PRICE = 2;
    const cmiOrder::MatchType AUTO_MATCH = 3;
  };
```

This interface is used for the match type provided by firms that match the internalized order.

```
  interface AuctionTypes
  {
    const cmiOrder::AuctionType AUCTION_INTERNALIZATION =1;
    const cmiOrder::AuctionType AUCTION_STRATEGY =2;  /*not currently used*/
```

```
    const cmiOrder::AuctionType AUCTION_REGULAR_SINGLE =3;  /* not currently used */
    const cmiOrder::AuctionType AUCTION_UNSPECIFIED = 0;    /* not currently used */
  };
```

This interface provides the types of auction codes that are supported.

```
interface AuctionStates
  {
    const cmiOrder::AuctionState STARTED = 1;
    const cmiOrder::AuctionState ENDED = 2;       /* not currently used */
    const cmiOrder::AuctionState PREMATURELY_ENDED = 3; /* not currently used */
    const cmiOrder::AuctionState ABORTED = 4;     /* not currently used */
  };
```

Constant codes interface that describe the state of the auction.

CMi Error Codes

```
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;  /* not
used */
     const exceptions::ErrorCode AUCTION_INACTIVE = 4130;  / *not used*/
     const exceptions::ErrorCode AUCTION_ENDED = 4140;
  };
```

```
interface DataValidationCodes  {
const exceptions::ErrorCode DUPLICATE_ID = 1000;
      const exceptions::ErrorCode INVALID_TIME = 1020;
      const exceptions::ErrorCode INCOMPLETE_QUOTE = 1030;
      const exceptions::ErrorCode INVALID_QUANTITY = 1040;
      const exceptions::ErrorCode INVALID_STRATEGY = 1060;
      const exceptions::ErrorCode INVALID_SPREAD = 1070;
      const exceptions::ErrorCode INVALID_USER = 1080;
      const exceptions::ErrorCode INVALID_PRODUCT = 1090;
      const exceptions::ErrorCode INVALID_SESSION = 1100;
      const exceptions::ErrorCode INVALID_STATE = 1110;
      const exceptions::ErrorCode PREFERENCE_PATH_MISMATCH = 1120;
      const exceptions::ErrorCode INVALID_ORDER_ID = 1130;
      const exceptions::ErrorCode LISTENER_ALREADY_REGISTERED = 1140;
      const exceptions::ErrorCode INVALID_SIDE = 1150;
      const exceptions::ErrorCode INVALID_PRICE = 1160;
      const exceptions::ErrorCode INVALID_UPDATE_ATTEMPT = 1170;
      const exceptions::ErrorCode INVALID_ORIGINATOR = 1180;
      const exceptions::ErrorCode INVALID_ACCOUNT = 1200;
      const exceptions::ErrorCode INVALID_EXECUTING_GIVEUP_FIRM = 1210;
      const exceptions::ErrorCode INVALID_CONTINGENCY_TYPE = 1220;
      const exceptions::ErrorCode INVALID_TIME_IN_FORCE = 1230;
      const exceptions::ErrorCode INVALID_POSITION_EFFECT = 1240;
      const exceptions::ErrorCode INVALID_ORIGIN_TYPE = 1250;
      const exceptions::ErrorCode INVALID_COVERAGE = 1260;
      const exceptions::ErrorCode INVALID_PRODUCT_TYPE = 1270;
      const exceptions::ErrorCode INVALID_ORDER_STATE = 1280;
      const exceptions::ErrorCode INVALID_ORDER_SOURCE = 1290;
      const exceptions::ErrorCode INVALID_BRANCH_SEQUENCE_NUMBER = 1300;
      const exceptions::ErrorCode MISSING_LISTENER = 1310;
      const exceptions::ErrorCode BUSINESS_DAY_NOT_STARTED = 1320;
      const exceptions::ErrorCode INVALID_FIELD_LENGTH = 1330;
      const exceptions::ErrorCode INVALID_STRATEGY_LEG = 1340;
      const exceptions::ErrorCode DUPLICATE_STRATEGY_LEG = 1350;
      const exceptions::ErrorCode INVALID_LEG_CONTINGENCY = 1360;
```

```
        const exceptions::ErrorCode INVALID_CANCEL_REQUEST = 1370;
        const exceptions::ErrorCode INVALID_VERSION = 1380;
        const exceptions::ErrorCode INVALID_LOGIN_MODE = 1390;
        const exceptions::ErrorCode GMD_LISTENER_ALREADY_REGISTERED = 1400;
        const exceptions::ErrorCode INVALID_TRADE_SOURCE = 1410;
        const exceptions::ErrorCode INVALID_TRADE_TYPE = 1420;
        const exceptions::ErrorCode NO_REMAINING_QUANTITY = 1430;
        const exceptions::ErrorCode INVALID_OPENING_REQUIREMENT = 1440;
        const exceptions::ErrorCode INVALID_PROCESS_NAME = 1450;
        const exceptions::ErrorCode INVALID_GROUP = 1460;

        // 1500 series for additions made for linkage support
        const exceptions::ErrorCode INVALID_EXCHANGE = 1500;
        const exceptions::ErrorCode INVALID_EXTENSIONS = 1510;
        const exceptions::ErrorCode INVALID_REJECT_REQUEST = 1520;

        // 1600 series for additions made to support internalized orders
        const exceptions::ErrorCode INVALID_MATCH_TYPE = 1600;
        const exceptions::ErrorCode INVALID_AUCTION_STATE = 1610;
        const exceptions::ErrorCode INVALID_AUCTION_ID = 1620;
        const exceptions::ErrorCode INTERNALIZATION_NOT_ALLOWED = 1630;
    };
```

These new constants are used to support internalized orders.

CMi Util

```
module cmiUtil
{

   typedef string VersionLabel;
   typedef short PriceType;
   typedef sequence <PriceType> PriceTypeSequence;
   typedef char  EntryType;
   typedef char  Side;
   typedef char  Source;
   typedef short UpdateStatusReason;
   typedef short ActivityReason;
```

```
typedef short QueryDirection;
typedef sequence <string> StringSequence;
typedef sequence <long> LongSequence;
typedef double PricingModelParameter;
typedef short ReportType;
typedef short OrderFlowDirection;
typedef short QueueAction;
typedef string Description;
typedef long Key;
typedef short LinkageMechanism;
typedef short SatisfactionOrderDisposition;
typedef short SatisfactionOrderRejectReason;
typedef short FillRejectReason;

struct DateStruct
{
   octet month;
   octet day;
   short year;
};
typedef sequence< DateStruct > DateStructSequence;

struct TimeStruct
{
   octet hour;
   octet minute;
   octet second;
   octet fraction;
};
typedef sequence< TimeStruct > TimeStructSequence;

struct DateTimeStruct
{
   cmiUtil::DateStruct date;
   cmiUtil::TimeStruct time;
};
typedef sequence< DateTimeStruct > DateTimeStructSequence;
```

```
struct PriceStruct
{
   cmiUtil::PriceType type;
   long whole;
   long fraction;
};
typedef sequence< PriceStruct > PriceStructSequence;

struct CallbackInformationStruct
{
   string subscriptionInterface;
   string subscriptionOperation;
   string subscriptionValue;
   string ior;
};

struct CboeIdStruct
{
   long highCboeId;
   long lowCboeId;
};

struct KeyValueStruct
{
   string key;
   string value;
};

typedef sequence <KeyValueStruct> KeyValueStructSequence;

struct KeyDescriptionStruct
{
  cmiUtil::Key key;
  cmiUtil::Description description;
};

typedef sequence <KeyDescriptionStruct> KeyDescriptionStructSequence;
```

```
   struct OperationResultStruct
   {
      exceptions::ErrorCode errorCode;
      string errorMessage;
   };
   typedef sequence <OperationResultStruct> OperationResultStructSequence;


};
```

This struct and corresponding sequence is used to define generic operation results when CBOE does not throw an exception.

CAS Simulator Changes

- Changes to support Auctions
- Removed BOA examples. CBOE will no longer support BOA.

Document Changes

API-01

- No changes.

API-02

- New section for internalization and automated auction.
- Expanded the section on the V3 Quoting to include a more detailed description of Quote Update Control ID.
- CBOE *does* accept options orders with contingency "Stop Limit".
- Updated the Quote Thresholds. These are not new thresholds, but the documents were out of date. Hybrid (in session W_MAIN) thresholds are 20 Quotes (products) per Mass Quote message (all quotes per message must be for the same underlying stock or index), 100 Mass Quote or Quote message calls per user per one (1) second period, and 2000 total quotes (products) per user per three (3) second period. CFE_MAIN (CBOE Futures Exchange) and ONE_MAIN (OneChicago) thresholds are 4 Quotes (products) per Mass Quote message, 50 Mass Quote or Quote message calls per user per one (1) second period, 1000 total quotes (products) per user per five (5) second period.

API-03

- Added new interfaces based on this release.
- IntermarketQuery.showMarketableOrderBookAtPrice - Currently not implemented
- Added Preferred DPM information in the optionalData field. Firms that want to give one DPM priority in participating in a trade use this field by specifying P:EXCH.FIRM or P:FIRM;

API-04

- Added new interfaces and definitions based on this release.
- Added Preferred DPM information in the optionalData field. Firms that want to give one DPM priority in participating in a trade use this field by specifying P:EXCH.FIRM or P:FIRM;
- IntermarketQuery.showMarketableOrderBookAtPrice - Currently not implemented

- getBookDepth, getBookDepthDetails, subscribeBookDepthForClassV2, subscribeBookDepthForProductV2, subscribeBookDepthUpdateForClassV2, subscribeBookDepthUpdateForProductV2 are all not allowed for classes in the W_MAIN session.
- Enhanced the descriptions for ListingStates.INACTIVE, OBSOLETE, and UNLISTED

API-05

- Removed the OMNI ORB and Visibroker BOA documentation. BOA examples will no longer be supported.

API-06

- No changes

API-07

- In W_MAIN, a user may only send getBookDepth once per 10 minutes per user for all options in the W_MAIN session.
- subscribeBookDepthForClassV2, subscribeBookDepthForProductV2, subscribeBookDepthUpdateForClassV2, subscribeBookDepthUpdateForProductV2 are all not allowed for classes in the W_MAIN session and are all not allowed for new development.
- getBookDepth and getMarketDataHistoryByTime are both allowed for new development (they were previously labeled as not allowed in the documents).

CAS-01

- No changes

CAS-02

- New simulator methods for internalization and auctions

Test Plan Changes

CMi Phase 2 Test Plans

- W_MAIN user ID format: ABC; CFE_MAIN user ID format: ABC_CFE, AB1_CFE; OneChicago user ID format: ABC, AB1, ABC1

3a, Security Definition Test Plan

- No changes

3b, Market Data Test Plan

- No changes

3c, Quote Test Plan - Hybrid-ONE-CFE (includes 3j, Hybrid Sections and 3L, CFE Supplemental Tests)

- No changes

3e, W_MAIN-ONE-CFE Order Test Plan (includes 3k, CFE Supplemental Tests)

- Added a Sending Internalized Orders section.
- Added an Auction section.
- Enhanced order description in first step GO.3 saying:
  Orders of origin "M" and "N" must set CMi.OptionalData or FIX tag 9324 to
  "`M:QAB ABC123LLL`"
  where QAB is the account, ABC123 is the subaccount, and LLL is the optional MM originator acronym.

3f, Clearing Firm, Duplicate Message Test Plan

- No changes

3g, Strategy Quote Test Plan - ONE-CFE

- No changes

3i, W_MAIN-ONE-CFE Strategy Order Test Plan

- No changes

3m, Stock Trading On CBOEdirect (STOC) Order Test Plan

- No changes

3n, Stock Trading On CBOEdirect (STOC) Quote Test Plan

- No changes

3o, Stock Trading On CBOEdirect (STOC) DPM Administrative Test Plan

- No changes

Phase 4 Test Plan

- No changes