CBOE Application Programming Interface

CBOE API Version 9.0 - Release Notes

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

CBOE PROPRIETARY INFORMATION

14 January 2011

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 9.0. Version 9.0 supports new CMi interfaces (CMi V9), 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 V9.0 Highlights

The sections below discuss the upcoming CBOEdirect software release that introduces the new CMi V9 interfaces. The CMi V9 interfaces support a new mechanism for order entry.

CMi V9 Interfaces

The CMi V9 interfaces support a new mechanism for entering orders that makes order entry light and rapid. CMi V9 provides an order entry interface that customers can use in lieu of quotes to take the added advantages of tiered quoting as well as one sided quoting.

Light order entry is accomplished by minimizing the order message size and by lessening the order status reports sent to the CMi user. CMi users, using the CMi V9 interfaces, will not receive NEW and Cancel reports. Instead, order details will be supplied as part of the return struct.

Session Management

The new CMi V9 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 UserSessionManagerV9. This reference can be obtained through logon using the UserAccessV9 interface. The new logon method takes exactly the same parameters as the old CMi's logon method and returns a newly enhanced SessionManagerV9. The UserAccessV9 object will be made available as an alternate IOR link on the HTTP port that the CAS is publishing on.

```
interface UserAccessV9
   {
         UserSessionManagerV9 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 UserSessionManagerV9 : cmiV8::UserSessionManagerV8
        {
            cmiV9::OrderEntry   getOrderEntryV9()
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::AuthenticationException,
            exceptions::NotFoundException
            );
      };
```

Light Order Entry

A light order is generated by calling the acceptLightOrder method in the cmiV7:OrderEntry interface. The acceptLightOrder method takes the cmiOrder:LightOrderEntryStruct as an argument and returns the cmiOrder:LightOrderResultStruct on successful entry. A NEW report will not be generated. Light orders support only simple and IOC orders.

```
    interface OrderEntry: cmiV7::OrderEntry
        {
            cmiOrder:: LightOrderResultStruct acceptLightOrder(
              in cmiOrder:: LightOrderEntryStruct anOrder)
                raises(
                  exceptions::SystemException,
                  exceptions::CommunicationException,
                  exceptions::AuthorizationException,
                  exceptions::DataValidationException,
                  exceptions::NotAcceptedException,
                  exceptions::TransactionFailedException,
                  exceptions::AlreadyExistsException
```

```
                        );

module cmiOrder
        {
        struct LightOrderEntryStruct
                  {
                    string branch;
                    long branchSequenceNumber;
                    long originalQuantity;
                    double Price;
                    cmiProduct::ProductKey productKey;
                    cmiUtil::Side side;
                    cmiOrder::PositionEffect positionEffect;
                    cmiOrder::Coverage coverage;
                    boolean isNBBOProtected;
                    boolean isIOC;
                    cmiOrder::OriginType orderOriginType;
                    cmiUser::Exchange cmtaExchange;
                    string cmtaFirmNumber;
                    string pdpm;
                    string userAssignedId;
                    cmiSession::TradingSessionName activeSession;
                  };

        struct LightOrderResultStruct
          {
            string branch;
            long branchSequenceNumber;
            long orderHighId;
            long orderLowId;
            cmiUtil::Side side;
            long leavesQuantity;
            long tradedQuantity;
            long cancelledQuantity;
            cmiUtil::ActivityReason reason;
            cmiUtil::DateTimeStruct time;
          };
           typedef sequence <LightOrderResultStruct>
      LightOrderResultStructSequence;
```

Cancel a Light Order

Light orders can be canceled using the methods: acceptLightOrderCancelRequest or acceptLightOrderCancelRequestById. A valid user assigned cancel ID, branch, branch sequence number, product key and active session is required to cancel a Light order. A successful cancel returns the cmiOrder:LightOrderResultStruct. A Cancel report will not be generated.

Light orders can be canceled only through the CMi V9 interfaces. Attempting to cancel a Light order through the existing CMi cancel interface will result in a cancel reject. In addition, the CMi V9 cancel interface is used to cancel only Light orders. Cancels targeting other orders will be rejected. Cancel replace of Light orders is not supported.

```
cmiOrder:: LightOrderResultStruct acceptLightOrderCancelRequest(
                in string branch,
                in long branchSequenceNumber,
                in cmiProduct::ProductKey productKey,
                in cmiSession::TradingSessionName activeSession,
                in string userAssignedCancelId
                        )
                raises(
                  exceptions::SystemException,
                  exceptions::CommunicationException,
                  exceptions::AuthorizationException,
                  exceptions::DataValidationException,
                  exceptions::NotAcceptedException,
                  exceptions::TransactionFailedException
                );

cmiOrder:: LightOrderResultStruct acceptLightOrderCancelRequestById(
                in long orderHighId,
                in long orderLowId,
                in cmiProduct::ProductKey productKey,
                    in cmiSession::TradingSessionName activeSession,
                    in string userAssignedCancelId
                              )
                    raises(
                       exceptions::SystemException,
                       exceptions::CommunicationException,
                       exceptions::AuthorizationException,
                       exceptions::DataValidationException,
                       exceptions::NotAcceptedException,
                       exceptions::TransactionFailedException
                    );
      };
```

Light Order Error Messages

CMi users need to be explicitly enabled to send Light orders by the CBOE Help Desk. Light orders are enabled at a user acronym level. Users that attempt to enter Light order but are not enabled or setup correctly will receive the following error messages.

```
interface DataValidationCodes
   const exceptions::ErrorCode INVALID_USER_ID_FOR_LIGHT_ORDERS = 1950;
   const exceptions::ErrorCode INVALID_ORIGIN_TYPE_FOR_LIGHT_ORDERS = 7200;


interface AuthorizationCodes
   const exceptions::ErrorCode USER_NOT_ENABLED_FOR_LIGHT_ORDERS = 7051;
```

New Activity Reason Message

In the event of a system failover, CMi users will receive an Activity Reason message, **const cmiUtil::ActivityReason QUOTE_UPDATES_REQUESTED = 25.** This message notifies the user that their quotes have been deleted from the system and requests that the user re-enter their quotes.

IDL Interfaces

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

```
module cmiV9

        interface OrderEntry: cmiV7::OrderEntry
        {
                cmiOrder:: LightOrderResultStruct acceptLightOrder(
                  in cmiOrder:: LightOrderEntryStruct anOrder)
                        raises(
                          exceptions::SystemException,
                          exceptions::CommunicationException,
                          exceptions::AuthorizationException,
                          exceptions::DataValidationException,
                          exceptions::NotAcceptedException,
                          exceptions::TransactionFailedException,
                          exceptions::AlreadyExistsException
                        );

                cmiOrder:: LightOrderResultStruct acceptLightOrderCancelRequest(
                        in string branch,
                        in long branchSequenceNumber,
                        in cmiProduct::ProductKey productKey,
                        in cmiSession::TradingSessionName activeSession,
                        in string userAssignedCancelId
                                )
                        raises(
                          exceptions::SystemException,
                          exceptions::CommunicationException,
                          exceptions::AuthorizationException,
                          exceptions::DataValidationException,
                          exceptions::NotAcceptedException,
                          exceptions::TransactionFailedException
                        );
```

```
            cmiOrder:: LightOrderResultStruct acceptLightOrderCancelRequestById(
                    in long orderHighId,
                    in long orderLowId,
                    in cmiProduct::ProductKey productKey,
                    in cmiSession::TradingSessionName activeSession,
                    in string userAssignedCancelId
                            )
                    raises(
                       exceptions::SystemException,
                       exceptions::CommunicationException,
                       exceptions::AuthorizationException,
                       exceptions::DataValidationException,
                       exceptions::NotAcceptedException,
                       exceptions::TransactionFailedException
                    );
    };

interface UserSessionManagerV9 : cmiV8::UserSessionManagerV8
    {
            cmiV9::OrderEntry   getOrderEntryV9()
    raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::AuthenticationException,
            exceptions::NotFoundException
            );

};

interface UserAccessV9
{
    UserSessionManagerV9 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
                        );
    };
};

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 short OrderMaintenanceType;
          typedef char OrderType;

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

```
#pragma use_factory_for_struct ON
    struct OrderContingencyStruct
    {
        cmiOrder::ContingencyType type;
        cmiUtil::PriceStruct price;
        long volume;
    };
#pragma use_factory_for_struct OFF

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

#pragma use_factory_for_struct OFF

    typedef sequence <OrderIdStruct> OrderIdStructSequence;

    struct ORDOrderStruct
    {
        OrderIdStruct orderId;
        cmiOrder::OrderState state;
        long bookedQuantity;
    };
    typedef sequence <ORDOrderStruct> ORDOrderStructSequence;

    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 LegOrderEntryStructV2
    {
        cmiOrder::LegOrderEntryStruct legOrderEntry;
        cmiUtil::Side side;
              string extensions;
    };
    typedef sequence <LegOrderEntryStructV2> LegOrderEntryStructV2Sequence;


    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;


#pragma use_factory_for_struct ON
    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;
    };
#pragma use_factory_for_struct OFF
    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;


    struct OrderResultStructV2
    {
        cmiOrder::OrderStruct order;
        cmiUtil::OperationResultStruct result;
    };
    typedef sequence <OrderResultStructV2> OrderResultStructV2Sequence;


    struct InternalizationOrderResultStructV2
    {
        cmiOrder::OrderResultStructV2 primaryOrderResult;
        cmiOrder::OrderResultStructV2 matchOrderResult;
    };
    typedef sequence <InternalizationOrderResultStructV2>
InternalizationOrderResultStructV2Sequence;


    struct CrossOrderStruct
    {
              cmiOrder::OrderStruct buySideOrder;
        cmiOrder::OrderStruct sellSideOrder;
};
struct LightOrderEntryStruct
{
    string branch;
    long branchSequenceNumber;
    long originalQuantity;
    double Price;
    cmiProduct::ProductKey productKey;
    cmiUtil::Side side;
    cmiOrder::PositionEffect positionEffect;
    cmiOrder::Coverage coverage;
    boolean isNBBOProtected;
    boolean isIOC;
    cmiOrder::OriginType orderOriginType;
    cmiUser::Exchange cmtaExchange;
    string cmtaFirmNumber;
    string pdpm;
    string userAssignedId;
    cmiSession::TradingSessionName activeSession;
};

struct LightOrderResultStruct
{
    string branch;
    long branchSequenceNumber;
    long orderHighId;
    long orderLowId;
    cmiUtil::Side side;
    long leavesQuantity;
    long tradedQuantity;
    long cancelledQuantity;
    cmiUtil::ActivityReason reason;
    cmiUtil::DateTimeStruct time;
};
  typedef sequence <LightOrderResultStruct> LightOrderResultStructSequence;
```

```
        };

module cmiConstants

    interface OrderOrigins
        {
        const cmiOrder::OriginType PRINCIPAL_ACTING_AS_AGENT = 'A';
        const cmiOrder::OriginType BROKER_DEALER = 'B';
        const cmiOrder::OriginType CUSTOMER = 'C';    //CTI Equivalent - Non Member, Customer
    Segregated Account
        const cmiOrder::OriginType CUSTOMER_FBW = 'D';
        const cmiOrder::OriginType CTI1Origin2 = 'E';       //Member, House Account
        const cmiOrder::OriginType FIRM = 'F';   //CTI Equivalent - Firm Trader, House Account
        const cmiOrder::OriginType CTI3Origin1 = 'G';    //User Proxy for trader, Customer Segregated
    Account
        const cmiOrder::OriginType CTI3Origin2 = 'H';           //User Proxy for trader, House Account
        const cmiOrder::OriginType MARKET_MAKER_IN_CROWD = 'I';
        const cmiOrder::OriginType FIRM_FBW_ICM = 'J';
        const cmiOrder::OriginType BROKER_DEALER_FBW_ICM = 'K';
        const cmiOrder::OriginType FIRM_FBW_NON_CUSTOMER = 'L';
        const cmiOrder::OriginType MARKET_MAKER = 'M';
        const cmiOrder::OriginType MARKET_MAKER_AWAY = 'N';
        const cmiOrder::OriginType CTI4Origin2 = 'O';     //Non Member, House Account
        const cmiOrder::OriginType PRINCIPAL = 'P';
        const cmiOrder::OriginType CTI1Origin5 = 'Q';     //Member, SIPC Protected Account
        const cmiOrder::OriginType CTI3Origin5 = 'R';    //User Proxy for trader, SIPC Protected
    Account
        const cmiOrder::OriginType SATISFACTION = 'S';
        const cmiOrder::OriginType CTI4Origin5 = 'T';      //Non Member, SIPC Protected Account
        const cmiOrder::OriginType M_N_Y_FBW = 'U';  //Market Maker, Non-CBOE Member,
    Specialist Away
        const cmiOrder::OriginType CTI1Origin1 = 'V';   //Member, Customer Segregated Account
        const cmiOrder::OriginType BROKER_DEALER_FBW_NON_CUSTOMER = 'W';
        const cmiOrder::OriginType CUSTOMER_BROKER_DEALER = 'X';
        const cmiOrder::OriginType UNDERLY_SPECIALIST = 'Y';
        const cmiOrder::OriginType N_Y_FBW = 'Z';    //Non-CBOE Member & Secialist Away. Now U
    is only for MarketMaker
```

```
        const cmiOrder::OriginType ITS_POR = 'a';    //Order on behalf of ITS PreOpening Response
    from away exchange
```

const cmiOrder::OriginType MANUAL_QUOTE_ORDER = 'K';   //Used by Manual Quote orders from PAR client

```
          };


    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;
```

//Added for SPX Linkage Fish and Return Functionality

const cmiTraderActivity::ActivityType MANUAL_ORDER_FR   = 720;

const cmiTraderActivity::ActivityType MANUAL_ORDER_FR_TIMEOUT = 721;

const cmiTraderActivity::ActivityType MANUAL_ORDER_FR_TIMEOUT_FAILURE =722;

```
interface ActivityReasons
   {
       const cmiUtil::ActivityReason NOTHING_DONE = 1;
       const cmiUtil::ActivityReason USER = 2;
       const cmiUtil::ActivityReason SYSTEM = 3;
       const cmiUtil::ActivityReason LOST_CONNECTION = 4;
       const cmiUtil::ActivityReason INSUFFICIENT_QUANTITY = 5;
       const cmiUtil::ActivityReason SPECIAL_ADJUSTMENT = 6;
       const cmiUtil::ActivityReason QRM_REMOVED = 7;
       const cmiUtil::ActivityReason INSUFFICIENT_QUANTITY_BUY_SIDE  = 8;
       const cmiUtil::ActivityReason INSUFFICIENT_QUANTITY_SELL_SIDE  = 9;
       const cmiUtil::ActivityReason QUOTE_UPDATE_CONTROL =10;
       // acceptServerFailure event would have following reason
       const cmiUtil::ActivityReason FAILOVER= 11;
       const cmiUtil::ActivityReason QUOTE_IN_TRIGGER =12;
       const cmiUtil::ActivityReason INVALID_SESSION_ID =13;
       const cmiUtil::ActivityReason SAL_IN_PROGRESS = 14;
       const cmiUtil::ActivityReason CROSS_IN_PROGRESS = 15;
       const cmiUtil::ActivityReason INVALID_NBBO = 16;
       const cmiUtil::ActivityReason NOT_WITHIN_NBBO = 17;
       const cmiUtil::ActivityReason TRADE_THROUGH_CBOE = 18;
       const cmiUtil::ActivityReason INSUFFICIENT_CUSTOMER_ORDER_QUANTITY = 19;
       const cmiUtil::ActivityReason INSUFFICIENT_CROSS_ORDER_SIZE = 20;
       const cmiUtil::ActivityReason INSUFFICIENT_CROSS_ORDER_DOLLAR_AMOUNT = 21;
       const cmiUtil::ActivityReason SELL_SHORT_RULE_VIOLATION = 22;
       // acceptUserActivityTimeout (UIM) event would have the following reason:
       const cmiUtil::ActivityReason NO_USER_ACTIVITY = 23;
       const cmiUtil::ActivityReason CANCEL_ON_RSS = 24;

       // acceptServerFailure will have the following reason code in case of CDX fast failover
       const cmiUtil::ActivityReason QUOTE_UPDATES_REQUESTED = 25;

       // The following are used for Linkage

       const cmiUtil::ActivityReason BROKER_OPTION = 100;
       const cmiUtil::ActivityReason CANCEL_PENDING = 101;
       const cmiUtil::ActivityReason CROWD_TRADE = 102;
       const cmiUtil::ActivityReason DUPLICATE_ORDER = 103;
       const cmiUtil::ActivityReason EXCHANGE_CLOSED = 104;
       const cmiUtil::ActivityReason GATE_VIOLATION = 105;
       const cmiUtil::ActivityReason INVALID_ACCOUNT = 106;
       const cmiUtil::ActivityReason INVALID_AUTOEX_VALUE = 107;
       const cmiUtil::ActivityReason INVALID_CMTA = 108;
       const cmiUtil::ActivityReason INVALID_FIRM = 109;
       const cmiUtil::ActivityReason INVALID_ORIGIN_TYPE = 110;
       const cmiUtil::ActivityReason INVALID_POSITION_EFFECT = 111;
       const cmiUtil::ActivityReason INVALID_PRICE = 112;
       const cmiUtil::ActivityReason INVALID_PRODUCT = 113;
       const cmiUtil::ActivityReason INVALID_PRODUCT_TYPE = 114;
       const cmiUtil::ActivityReason INVALID_QUANTITY = 115;
       const cmiUtil::ActivityReason INVALID_SIDE = 116;
       const cmiUtil::ActivityReason INVALID_SUBACCOUNT = 117;
       const cmiUtil::ActivityReason INVALID_TIME_IN_FORCE = 118;
       const cmiUtil::ActivityReason INVALID_USER = 119;
       const cmiUtil::ActivityReason LATE_PRINT = 120;
       const cmiUtil::ActivityReason NOT_FIRM = 121;
       const cmiUtil::ActivityReason MISSING_EXEC_INFO = 122;
       const cmiUtil::ActivityReason NO_MATCHING_ORDER = 123;
       const cmiUtil::ActivityReason NON_BLOCK_TRADE = 124;
       const cmiUtil::ActivityReason NOT_NBBO = 125;
```

```
    const cmiUtil::ActivityReason COMM_DELAYS = 126;
    const cmiUtil::ActivityReason ORIGINAL_ORDER_REJECTED = 127;
    const cmiUtil::ActivityReason OTHER = 128;
    const cmiUtil::ActivityReason PROCESSING_PROBLEMS = 129;
    const cmiUtil::ActivityReason PRODUCT_HALTED = 130;
    const cmiUtil::ActivityReason PRODUCT_IN_ROTATION = 131;
    const cmiUtil::ActivityReason STALE_EXECUTION = 132;
    const cmiUtil::ActivityReason STALE_ORDER = 133;
    const cmiUtil::ActivityReason ORDER_TOO_LATE = 134;
    const cmiUtil::ActivityReason TRADE_BUSTED = 135;
    const cmiUtil::ActivityReason TRADE_REJECTED = 136;
    const cmiUtil::ActivityReason ORDER_TIMEOUT = 141;
    const cmiUtil::ActivityReason REJECTED_LINKAGE_TRADE  = 170;
    const cmiUtil::ActivityReason SATISFACTION_ORD_REJ_OTHER  = 171;
    const cmiUtil::ActivityReason PRODUCT_SUSPENDED = 172;

    // Currently used for TPF linkage; in future may be used for CBOEdirect
    const cmiUtil::ActivityReason UNKNOWN_ORDER = 137;
    const cmiUtil::ActivityReason INVALD_EXCHANGE = 138;
    const cmiUtil::ActivityReason TRANSACTION_FAILED = 139;
    const cmiUtil::ActivityReason NOT_ACCEPTED = 140;

    // Used for linkage when cancel reason is not provided (could be user cancel or cancel remaining)
    const cmiUtil::ActivityReason AWAY_EXCHANGE_CANCEL = 199;

                // Force Cancel Orders due to fallback
                const cmiUtil::ActivityReason CANCEL_ON_FALLBACK = 800;

    // Linkage Business Message Reject codes
    const cmiUtil::ActivityReason LINKAGE_CONDITIONAL_FIELD_MISSING = 900;
    const cmiUtil::ActivityReason LINKAGE_EXCHANGE_UNAVAILABLE = 901;
    const cmiUtil::ActivityReason LINKAGE_INVALID_MESSAGE = 902;
    const cmiUtil::ActivityReason LINKAGE_INVALID_DESTINATION = 903;
    const cmiUtil::ActivityReason LINKAGE_INVALID_PRODUCT = 904;
    const cmiUtil::ActivityReason LINKAGE_SESSION_REJECT = 905;
};
```

```
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";
const ExtensionField DAIM_MATCH_INDICATOR = "dmatch";
```

// These extensions are added for Manual Quote Liked Order
const ExtensionField MANUAL_ORDER_ENABLE = "manualOrder";
const ExtensionField MANUAL_QUOTE_LOCATION_ID = "locationId";
const ExtensionField MANUAL_QUOTE_IP_ADDRESS = "ipAddress";

```
      const ExtensionField MANUAL_QUOTE_PAR_ID = "parId";

  };

module cmiIntermarketMessages
        {

          typedef char OrderBookTradableType;
          typedef short FillRejectReason;
          typedef short HandlingInstruction;
          typedef short ExchangeMarketInfoType;
          typedef short OrderBookStatus;
          typedef short PreOpeningIndicationType;
          typedef short AlertType;
          typedef string AlertResolution;
          typedef sequence <cmiOrder::OrderIdStruct > OrderIdSequence;

          struct ExchangeMarketStruct
          {
            cmiIntermarketMessages::ExchangeMarketInfoType marketInfoType;
            cmiUtil::PriceStruct bestBidPrice;
            cmiMarketData::ExchangeVolumeStructSequence bidExchangeVolumes;
            cmiUtil::PriceStruct bestAskPrice;
            cmiMarketData::ExchangeVolumeStructSequence askExchangeVolumes;
          };

          typedef sequence <ExchangeMarketStruct> ExchangeMarketStructSequence;

          struct ExchangeMarketStructV2
          {
            cmiIntermarketMessages::ExchangeMarketInfoType marketInfoType;
            cmiUtil::PriceStruct bestBidPrice;
            cmiMarketData::ExchangeVolumeStructSequence bidExchangeVolumes;
            cmiUtil::PriceStruct bestAskPrice;
            cmiMarketData::ExchangeVolumeStructSequence askExchangeVolumes;
            long long time;
```

```
    char usedForTradeThrough;
};

typedef sequence <ExchangeMarketStructV2> ExchangeMarketStructV2Sequence;

struct HeldOrderStruct
{
   cmiOrder::OrderStruct order;
   cmiIntermarketMessages::ExchangeMarketStructSequence currentMarketBest;
};

typedef sequence <HeldOrderStruct> HeldOrderStructSequence;

struct HeldOrderDetailStruct
{
    cmiProduct::ProductNameStruct productInformation;
    cmiUtil::UpdateStatusReason statusChange;
    cmiIntermarketMessages::HeldOrderStruct heldOrder;
};

typedef sequence <HeldOrderDetailStruct> HeldOrderDetailStructSequence;

struct OrderReminderStruct
{
    cmiOrder::OrderIdStruct reminderId;
    string reminderReason;
    cmiUtil::DateTimeStruct timeSent;
};

struct HeldOrderCancelRequestStruct
{
    cmiUtil::CboeIdStruct cancelReqId;
    cmiOrder::CancelRequestStruct cancelRequest;
};
typedef sequence <HeldOrderCancelRequestStruct> HeldOrderCancelRequestStructSequence;
```

```
struct HeldOrderCancelReportStruct
{
   HeldOrderDetailStruct heldOrderDetail;
   cmiUtil::CboeIdStruct cancelReqId;
   cmiOrder::CancelReportStruct cancelReport;
};

struct HeldOrderFilledReportStruct
{
   HeldOrderDetailStruct heldOrderDetail;
   cmiOrder::FilledReportStructSequence filledReport;
};

struct CurrentIntermarketBestStruct {
    cmiUser::Exchange exchange;
    cmiSession::ProductState marketCondition;
    cmiUtil::PriceStruct bidPrice;
    long bidVolume;
    cmiUtil::PriceStruct askPrice;
    long askVolume;
    cmiUtil::TimeStruct sentTime;
};
typedef sequence <CurrentIntermarketBestStruct> CurrentIntermarketBestStructSequence;

struct CurrentIntermarketStruct
{
    cmiProduct::ProductKeysStruct productKeys;
    CurrentIntermarketBestStructSequence otherMarketsBest;
};
typedef sequence <CurrentIntermarketStruct> CurrentIntermarketStructSequence;

struct FillRejectStruct
{
    cmiUtil::CboeIdStruct tradeId;
    cmiOrder::OrderStruct order;
    long transactionSequenceNumber;
```

```
        cmiUtil::FillRejectReason rejectReason;
        string extensions;
        string text;
    };
    typedef sequence <FillRejectStruct> FillRejectStructSequence;

    struct OrderFillRejectStruct
    {
        cmiOrder::OrderDetailStruct rejectedFillOrder;
        cmiIntermarketMessages::FillRejectStructSequence fillRejectReports;
    };
    typedef sequence <OrderFillRejectStruct> OrderFillRejectStructSequence;

    struct FillRejectRequestStruct
    {
        cmiOrder::OrderIdStruct orderId;
        cmiUtil::ActivityReason rejectReason;
        string fillReportExtensions;
        long tradedQuantity;
        cmiUtil::PriceStruct tradePrice;
    };

    struct PreOpeningIndicationPriceStruct
    {
        cmiIntermarketMessages::PreOpeningIndicationType preOpenType;
        cmiOrder::OriginType preOpenOriginType;
        cmiUtil::PriceStruct lowOpeningPrice;
        cmiUtil::PriceStruct highOpeningPrice;
        cmiUtil::Side side;
        long principalQuantity;
    };
    typedef  sequence <PreOpeningIndicationPriceStruct>
PreOpeningIndicationPriceStructSequence;

    struct PreOpeningResponsePriceStruct
    {
        cmiOrder::OrderState orderState;
        cmiUtil::Side side;
        long principalQuantity;
        long agencyQuantity;
        cmiUtil::PriceStruct responsePrice;
    };
    typedef  sequence <PreOpeningResponsePriceStruct>
PreOpeningResponsePriceStructSequence;

    struct AdminStruct
    {
        cmiAdmin::MessageStruct messageStruct;
        string userAssignedId;
        cmiProduct::ProductKey productKey;
        cmiUser::Exchange sourceExchange;
        cmiUser::Exchange destinationExchange;
    };
    typedef  sequence <AdminStruct> AdminStructSequence;

    struct AlertHdrStruct{        //this information is required in all alert kinds
         cmiUtil::CboeIdStruct alertId;
         cmiUtil::DateTimeStruct alertCreationTime;
         cmiIntermarketMessages::AlertType alertType;
         cmiSession::TradingSessionName sessionName;
         string hdrExtensions;
    };

    struct AlertStruct{        // these are alerts sent to the TFL related to NBBO Agent
         AlertHdrStruct alertHdr;
         AlertResolution resolution;
         string comments;
         cmiOrder::OrderIdStruct orderId;
         string nbboAgentId;        // rename Userid to nbboAgentId for clarity
         string updatedById;        // new field to keep track of who updated the field last
         cmiUtil::CboeIdStruct tradeId;
         cmiProduct::ProductKeysStruct productKeys;
        cmiIntermarketMessages::ExchangeMarketStructSequence exchangeMarket;
        boolean cboeMarketableOrder; // If order was marketable against CBOE BBBO
        string extensions;
    };
    typedef sequence <AlertStruct> AlertStructSequence;

    struct AlertStructV2{        // these are alerts sent to the TFL related to NBBO Agent
        AlertHdrStruct alertHdr;
        AlertResolution resolution;
        string comments;
        cmiOrder::OrderIdStruct orderId;
        string nbboAgentId;        // rename Userid to nbboAgentId for clarity
        string updatedById;        // new field to keep track of who updated the field last
        cmiUtil::CboeIdStruct tradeId;
        cmiProduct::ProductKeysStruct productKeys;
        cmiIntermarketMessages::ExchangeMarketStructV2Sequence exchangeMarket;
        boolean cboeMarketableOrder; // If order was marketable against CBOE BBBO
        string extensions;
        cmiUtil::PriceStruct tradedThroughPrice;
        long tradedThroughQuantity;
        cmiUtil::Side side;
        string orsId;

    };
    typedef sequence <AlertStructV2> AlertStructV2Sequence;

    struct SatisfactionAlertStruct{     // This is an alert that our book was traded through by a remote
exchange
       AlertHdrStruct alertHdr;
       long tradedThroughquantity;
       cmiUtil::PriceStruct tradedThroughPrice;
       cmiUtil::Side side;
       cmiMarketData::TickerStruct lastSale; // remote exchange last sale causing tradethrough
       OrderIdSequence tradedThroughOrders;  //our orders that were traded through
       string extensions;
    };
```

```
typedef sequence <SatisfactionAlertStruct> SatisfactionAlertStructSequence;

struct OrderBookStruct
{
    cmiOrder::OrderIdStruct orderId;
    long originalQuantity;
    long remainingQuantity;
    cmiProduct::ClassKey classKey;
    cmiProduct::ProductKey productKey;
    cmiProduct::ProductType productType;
    cmiUtil::Side side;
    cmiUtil::PriceStruct price;
    cmiOrder::TimeInForce timeInForce;
    cmiUtil::DateTimeStruct receivedTime;
    cmiOrder::OrderContingencyStruct contingency;
    cmiOrder::OriginType orderOriginType;
    cmiOrder::OrderState state;
    cmiOrder::NBBOProtectionType orderNBBOProtectionType;
    string optionalData;
    char tradableType;
};
typedef sequence <OrderBookStruct> OrderBookStructSequence;

struct OrderBookDetailPriceStruct
{
    OrderBookStructSequence orderInfo;
    cmiUtil::PriceStruct price;
};
typedef sequence <OrderBookDetailPriceStruct> OrderBookDetailPriceStructSequence;

struct BookDepthDetailedStruct
{
    cmiProduct::ProductKeysStruct productKeys;
    cmiSession::TradingSessionName sessionName;
    OrderBookDetailPriceStructSequence buyOrdersAtDifferentPrice;
    OrderBookDetailPriceStructSequence sellOrdersAtDifferentPrice;
```

```
            long transactionSequenceNumber;
        };
    };

module cmiErrorCodes
    {
        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_STRATEGY_RATIO = 1061;
            const exceptions::ErrorCode INVALID_SPREAD = 1070;
            const exceptions::ErrorCode INVALID_USER = 1080;
            const exceptions::ErrorCode INVALID_PRODUCT = 1090;
            const exceptions::ErrorCode INVALID_PRODUCT_CLASS = 1091;
            const exceptions::ErrorCode INVALID_REPORTING_CLASS = 1092;
            const exceptions::ErrorCode INVALID_PRODUCT_DESCRIPTION = 1093;
            const exceptions::ErrorCode INVALID_OPRA_MONTH_CODE = 1094;
            const exceptions::ErrorCode INVALID_PRICE_ADJUSTMENT = 1095;
            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 NO_WORKING_ORDER = 1135;
            const exceptions::ErrorCode LISTENER_ALREADY_REGISTERED = 1140;
            const exceptions::ErrorCode INVALID_SIDE = 1150;
            const exceptions::ErrorCode MISSING_SIDE_INDICATOR = 1151;
            const exceptions::ErrorCode INVALID_SIDE_INDICATOR = 1152;
            const exceptions::ErrorCode SIDE_INDICATOR_MISMATCH = 1153;
            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 MULTICLASS_STRATEGY_NOT_ALLOWED = 1345;
            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;
            const exceptions::ErrorCode INVALID_NAME = 1461;
            const exceptions::ErrorCode INVALID_THRESHOLD = 1462;
            const exceptions::ErrorCode INVALID_OPERATOR = 1463;
            const exceptions::ErrorCode INVALID_TRADE_REPORT_HANDLING_INSTRUCTION
    = 1464;
        const exceptions::ErrorCode INVALID_OPERATION_TYPE = 1475;
        // GroupService related section
            const exceptions::ErrorCode INVALID_GROUPELEMENT = 1474;
            const exceptions::ErrorCode INVALID_GROUPELEMENT_TYPE = 1465;
            const exceptions::ErrorCode INVALID_GROUPELEMENT_RELATIONSHIP = 1466;
```

```
    const exceptions::ErrorCode GROUPELEMENT_ALREADY_EXISTS = 1467;
    const exceptions::ErrorCode GROUPRELATIONSHIP_ALREADY_EXISTS = 1468;
    const exceptions::ErrorCode INVALID_USERID_REQUESTING_CANCEL = 1469;
    const exceptions::ErrorCode INVALID_WORKSTATION_ID = 1470;
    const exceptions::ErrorCode INVALID_USERID_LIST = 1471;
    const exceptions::ErrorCode ROOT_ALREADY_EXISTS = 1472;
    const exceptions::ErrorCode INVALID_GROUP_TYPE = 1473;

// 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;

const exceptions::ErrorCode INVALID_ID = 1530;
const exceptions::ErrorCode INVALID_POLICIES = 1531;
const exceptions::ErrorCode INVALID_LIMITS = 1532;
const exceptions::ErrorCode INVALID_TYPE = 1533;
const exceptions::ErrorCode INVALID_COUNT_BOUNDARIES = 1534;
const exceptions::ErrorCode INVALID_TIME_BOUNDARIES = 1535;

// 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;
const exceptions::ErrorCode INVALID_AUCTION_TYPE = 1640;
const exceptions::ErrorCode INVALID_OPTIONAL_DATA = 1650;

// 1700 series for additions made to support index hybrid feature
const exceptions::ErrorCode INVALID_CONTINGENCY_BOB_IORDER = 1700;
const exceptions::ErrorCode INVALID_CONTINGENCY_VIX_SETTLEMENT = 1701;

// 1710 series for  Cross Product
const exceptions::ErrorCode UNSUPPORTED_ORIGIN_TYPE = 1711;
const exceptions::ErrorCode UNDERLYING_LEG_NOT_LISTED_INSTOCK = 1712;
const exceptions::ErrorCode INVALID_RATIO_FOR_CROSS_PROD = 1713;
```

```
        const exceptions::ErrorCode INVALID_LEG_STOCK_PROD_STATE = 1714;
        const exceptions::ErrorCode INVALID_CLEARING_FIRM = 1715;

        //1800 series for manual price reporting
        const exceptions::ErrorCode END_OF_SALE = 1800;
        const exceptions::ErrorCode NOT_AN_OPENING_ONLY_TRADE = 1801;
        const exceptions::ErrorCode NO_TRADE_SO_FAR = 1802;
        const exceptions::ErrorCode NOT_AN_ONLY_TRADE = 1803;
        const exceptions::ErrorCode ONLY_OPENING_TRADE_SO_FAR = 1804;
        const exceptions::ErrorCode EITHER_LAST_SALE_OR_OPENING_TRADE = 1805;
        const exceptions::ErrorCode PRICE_NOT_EQUAL_TO_LAST_SALE = 1806;
        const exceptions::ErrorCode CANCELED_VOL_NOT_CUMMULATIVE_VOL = 1807;
        const exceptions::ErrorCode PRICE_NOT_EQUAL_TO_OPENING_PRICE = 1808;
        const exceptions::ErrorCode PRICE_GREATER_THAN_HIGH = 1809;
        const exceptions::ErrorCode PRICE_LESS_THAN_LOW = 1810;
        const exceptions::ErrorCode VOLUME_GREATER_THAN_CUMUMATIVE_VOL =
1811;

    // 1900 series for OHS Release One
            const exceptions::ErrorCode INVALID_CONTRA_BROKER = 1900;
        const exceptions::ErrorCode INVALID_CONTRA_FIRM = 1901;
        const exceptions::ErrorCode INVALID_EXECUTING_BROKER = 1903;
        const exceptions::ErrorCode INVALID_EXECUTING_BROKER_FIRM = 1904;
        const exceptions::ErrorCode INVALID_CABINET_ON_CXLRE = 1905;
        const exceptions::ErrorCode INVALID_NO_CURRENT_MARKET = 1906;
            const exceptions::ErrorCode INVALID_UPDATE_PRICE_REPORT = 1907;
            const exceptions::ErrorCode INVALID_NO_MATCHING_MDH = 1908;
            const exceptions::ErrorCode ORDER_REJECTED_ON_RSS = 1909;

            // light orders
            const exceptions::ErrorCode INVALID_USER_ID_FOR_LIGHT_ORDERS =
1950;

            // 2000 series for DirectedAIM
            const exceptions::ErrorCode INVALID_DIRECTED_AIM_TARGET_FIRM = 2000;
    };
```

```
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;

  // Short Sale
  const exceptions::ErrorCode SELL_SHORT_REJECT = 7100;
  const exceptions::ErrorCode INVALID_ORIGIN_TYPE_FOR_LIGHT_ORDERS = 7200;
    };

 interface AuthorizationCodes
    {
     const exceptions::ErrorCode USER_DISABLED = 7000;
     const exceptions::ErrorCode NOT_PERMITTED = 7001;
     const exceptions::ErrorCode INVALID_SESSION_ID = 7002;
     const exceptions::ErrorCode MAX_TIMEOUT_EXCEEDED = 7003;
     const exceptions::ErrorCode INVALID_PASSWORD = 7004;
     const exceptions::ErrorCode AUTHENTICATION_ERROR = 7005;
     const exceptions::ErrorCode INVALID_LOCATION = 7006;
     const exceptions::ErrorCode USER_NOT_ENABLED_FOR_LIGHT_ORDERS = 7051;
    };
```

Document Changes

API-01

- No changes

API-02

- Incorporated a new section to include the CMi V9 functionality.
- Added a section named "Special Considerations for Light Orders" that includes:
  - Cancel Replace of Light Orders are not supported.
  - No user query methods are supported for Light Orders.
  - Partial cancels are not supported.
  - The correspondentFirm will not be available in the case of Light Orders as it will be filled in with the userID internally. Moreover, because of this, the userIDs used for doing Light Orders may not be longer than 4 characters.
  - Origin codes that correspond to quote-like orders (e.g. "I") are not allowed for Light Orders.
  - Bust Reinstate of Light Orders is not supported.
  - UserAssignedId and UserAssignedCancelId fields are limited to 8 characters
  - Light Orders are supported for both options as well as stocks.
  - Cancel Requests for light orders are supported.
  - Market Orders are not allowed.
  - Light Orders are always treated as *DAY* orders.
  - The only contingency that is allowed is *IOC*.
  - Light Orders are not allowed on BOB classes.
  - Light Orders are not allowed on restricted series.
  - All Light Orders for a user will be canceled on logout.
  - Light Orders are not auction eligible. Also, Light Orders cannot be used to respond to auctions. However, Light Orders can possibly end an auction just like regular orders.
  - Light Orders are NBBO protected only if the NBBO protected flag is set to true in the incoming order.
  - Light Orders are never linked away. The order will be canceled if another exchange is at a better price and the NBBO protection is ON.
  - Light orders must be included in the opening and as part of the opening trades. If any remaining light orders would normally go through HALO, they still should as long as they are marked as NBBO protected. Light Orders that are not filled in HALO must

not be routed away; instead they must be cancelled. Light Orders those are not marked as NBBO protected must be cancelled rather than go through HALO.

API-03

- Added values for the new constants based on this release.
- Provided new class diagrams for CMi V9 interfaces.

API-04

- 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

- No changes

Simulator

- No changes

Test Plan Changes

- No changes