CONFIDENTIAL

CBOE Application Programming Interface

CBOE API Version 7.0 - Release Notes

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

CBOE PROPRIETARY INFORMATION

08 January 2010 (updated: 3/23/10)

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

Republished updated release notes on March 23, 2010 to reflect changes in cmiCallbackV5.

These release notes were initially published on January 2, 2010. Since then, new CMi V8 interfaces have been introduced and have been incorporated in this document. This document highlights upcoming changes in the new release of the CMi API, Version 7.0. Version 7.0 supports new CMi interfaces (CMi V7 and CMi V8), 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 V7.0 Highlights

The sections below discuss upcoming CBOEdirect software releases that introduce new CMi V7 interfaces. The CMi V7 interfaces support two new features; synchronous order entry changes and short sale marking functionality for orders and quotes. In addition, the upcoming releases present new CBSX cross order contingency types, instructions for Qualified Contingent Trade (QCT) orders and overall documentation changes.

CMi V7 Interfaces

Session Management

The new CMi V7 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 UserSessionManagerV7. This reference can be obtained through logon using the UserAccessV7 interface. The new logon method takes exactly the same parameters as the old CMi's logon method and returns a newly enhanced SessionManagerV7. The UserAccessV7 object will be made available as an alternate IOR link on the HTTP port that the CAS is publishing on.

```
interface UserAccessV7
  {
    UserSessionManagerV7 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
        );
};
```

Synchronous Order Entry

The CMi V7 interfaces support a new paradigm for entering orders that will make order entry a true synchronous call. Current order entry interface methods return order id information in an OrderIdStruct, and they result in asynchronous new order acknowledgement reports being delivered. Orders entered through the CMi V7 order entry methods will return a full OrderStruct (or some variation of an OrderStruct wrapper) containing all the pertinent order information, including the embedded OrderIdStruct, and will not result in a new order acknowledgement being delivered. Refer to the IDL Interfaces section of this document for CMi V7 interfaces.

New Order Entry methods

New V7 order entry methods have been added in this release. Both the NoAckV7 and the V7 for strategies methods provide the 'Side' field on the Strategy Orders to indicate possible short sale positions.

The NoAckV7 methods will differ in their return behavior from the current methods – these do not generate an asynchronous new order acknowledgement report. The new V7 methods retain current behavior, returning an asynchronous new order acknowledgement on the order entry. Behavior will remain the same for the currently existing order entry methods.

In addition, this release provides new methods for submission of new orders for strategies, cancel replaces for strategies and internalization orders for strategies.

Refer to the "IDL Interfaces" section of this document for CMi V7 interfaces.

New Quote Entry Methods

New V7 methods are provided in this release for submitting both single quotes and mass quotes. These methods use the new QuoteEntryStructV4, which allows users to submit the short sale position information. The V7 method will continue to maintain existing return behavior – the result of a successful mass quote submission will continue to be a ClassQuoteResultStructV3Sequence. Refer to the "IDL Interfaces" section of this document for CMi V7 interfaces.

Short Sale Marking

The CMi V7 interface will provide support for indicating short sale positions on all orders, quotes and mass quotes.

- For simple order entry, a short sale can be indicated using the existing 'Side' indicator on the OrderEntryStruct, cmiUtil::Side side.

```
module cmiOrder
{
    struct OrderEntryStruct
            {
               cmiUser::ExchangeFirmStruct executingOrGiveUpFirm;
               string branch;
```

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

- For Complex Order Entry, the new LegOrderEntryStructV2 now provides a similar ‘Side’ indicator to indicate the short sale of a particular leg.

```
module cmiOrder
{
    struct LegOrderEntryStructV2
        {
            cmiOrder::LegOrderEntryStruct legOrderEntry;
            cmiUtil::Side side;
             string extensions;
        };
```

- For quote entry, a similar ‘Side’ field is now also provided on the QuoteEntryStructV4 to enable the indication of the Short Sale on the individual quote, or on each of the quotes of the mass quote.

```
module cmiQuote
        struct QuoteEntryStructV4
            {
                cmiQuote::QuoteEntryStructV3 quoteEntryV3;
                cmiUtil::Side sellShortIndicator;
```

```
        string extensions;
    };
typedef sequence <QuoteEntryStructV4> QuoteEntryStructV4Sequence;
```

CBSX Cross Order Contingency Types

The late January 2010 software release of CBOEdirect will support three new cross order contingency types: (1) NEXT_DAY_CROSS, (2) TWO_DAY_CROSS and (3) CASH_CROSS. These contingency types are presented as CMi constants.

```
interface ContingencyTypes
  {
    const cmiOrder::ContingencyType CASH_CROSS = 32; // cash settlement cross
    const cmiOrder::ContingencyType NEXT_DAY_CROSS = 33; // Next day settlement cross
    const cmiOrder::ContingencyType TWO_DAY_CROSS  = 34; // Two day settlement cross
  };
```

These contingency types apply only to equity orders and the CBSX trading session. The new contingencies are not subject to the Reg NMS protections and are allowed to trade at a price of up to 4 decimal places. Regardless of the NBBO or CBSX Book, these contingency types should execute at the limit price of the orders.

Qualified Contingent Trade

The late January 2010 software release of CBOEdirect will give CMi users the ability to instruct CBOE to treat paired orders as Qualified Contingent Trade (QCT) orders. The objective is to allow hedged order pairs to cross immediately, ahead of any resting customer orders for QCT orders. In the optionalData field of the OrderEntryStruct, CMi users must enter A:AIQ as the first characters in the Primary Order to direct CBOE to treat the paired orders as QCT at the start of the AIM auction. There is no indication on the QCT Match Order indicating it is a QCT match except for the Primary Order information in the OptionalData field.

```
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;
};
```

CMi V8 Interfaces

CMi V8.0 module provides a new interface to report outage on a trading class during trading hours. The new service is based on a subscription mechanism. Subscription could be made on a trading group (which is a list of classes) or at a class level within a trading group. Once a group/class is marked down. CBOE will not accept any new orders/Cancel Replace/ Quotes on that class. However, order/quote cancels will be processed.

Session Management

The new CMi V8 interfaces will be an extension to the existing CMi on the existing CAS. A CMi user can gain access to "TradingClassStatusQuery" by getting a reference to the UserSessionManagerV8. This reference can be obtained through logon using the UserAccessV8 interface. The new logon method takes exactly the same parameters as the old CMi's logon method and returns the new SessionManagerV8. The UserAccessV8 object will be made available as an alternate IOR link on the HTTP port that the CAS is publishing on.

```
interface UserAccessV8
{
        UserSessionManagerV8 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 UserSessionManagerV8 : cmiV7::UserSessionManagerV7
        {
                TradingClassStatusQuery getTradingClassStatusQuery()
        raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::AuthenticationException,
                exceptions::NotFoundException
                );
        };
```

New Trading Class Status Query Interface

This new interface available through SessionManagerV8 provides the list off all Trading Groups and the list classes traded in each group. It also has two different subscriptions for Trading Class Status. The subscription methods take a callback to notify clients of any outage. There are four methods in TradingClassStatusQuery interface and a new CallbackV5 for clients to implement in this release.

```
interface TradingClassStatusQuery
{
        cmiProduct::ProductGroupSequence getProductGroups()
        raises(
                        exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::DataValidationException,
                exceptions::NotFoundException,
                exceptions::AuthorizationException
        );

        cmiProduct::ClassKeySequence getClassesForProductGroup(in
cmiProduct::ProductGroup productGroupName)
                raises(
                        exceptions::SystemException,
                        exceptions::CommunicationException,
                        exceptions::DataValidationException,
                        exceptions::NotFoundException,
                        exceptions::AuthorizationException
        );


        void subscribeTradingClassStatusForProductGroup(
        in cmiSession::TradingSessionName sessionName,
                        in cmiProduct::ProductGroupSequence roductGroupNames,
                        in cmiCallbackV5::CMITradingClassStatusQueryConsumer
                        clientListener)
```

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

        void subscribeTradingClassStatusForClasses(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKeySequence classKeys,
                        in cmiCallbackV5::CMITradingClassStatusQueryConsumer
                        clientListener)
        raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException
    };
};
```

Getting Groups/ Classes for Subscription:

The new Interface provides a method getProductGroups() which returns a list of all trading groups irrespective of trading session.
The method "getClassesForProductGroup" takes a group name as parameter and returns the list classes traded in that group.

Subscription based on Groups:

Subscriptions could be made on the group level. A group will have at least one or more trading classes. The method "subscribeTradingClassStatusForProductGroup" helps you do this. The method take in the Trading Session(like W_MAIN), the list group Name and a callback to notify end clients.

Subscription based on ClassKeys:

Subscriptions could also be made as granular as at the class level. A classes may or may not have products/series defined under them. The method "subscribeTradingClassStatusForClasses" helps you do this. The method take in the Trading Session(like W_MAIN), the list class keys and a callback to notify end clients.

Note: There are no unsubscribing methods in the new interface. All trading class status subscriptions are cancelled when user logs out.

CmiCallBackV5 Interface

There are two methods in this new interface. Based on the subscription type (group or class) appropriate methods are called. This release introduces new CmiConstants to notify users of the trading class status.

```
interface CMITradingClassStatusQueryConsumer {
        void acceptTradingClassStatusUpdateforProductGroups(
```

```
        in cmiProduct::ProductGroupSequence listOfProductGroups,
        in cmiUtil::TradingClassStatusIndicator status);

        void acceptTradingClassStatusUpdateforClasses(
        in cmiProduct::ClassKeySequence listOfClasses,
        in cmiUtil::TradingClassStatusIndicator status);
        };
```

New CMiConstants Interface

These two constant are send out as trading class status. Once a trading class closed due to outage. CBOE will not accept any new Orders, Cancel Replace or Quotes for that group or class. However Order/Quote/System generated Cancels will be processed.

```
interface TradingClassStatusIndicators
        {
        const short CLOSED_OUTAGE          = 1;
        const short OPEN_AFTER_OUTAGE   = 4;
        };
```

IDL Interfaces

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

```
module cmiV7
{
  interface OrderEntry: cmiV5::OrderEntry
  {
    cmiOrder::OrderStruct acceptOrderNoAckV7(
      in cmiOrder::OrderEntryStruct anOrder)
        raises(
          exceptions::SystemException,
          exceptions::CommunicationException,
          exceptions::AuthorizationException,
          exceptions::DataValidationException,
          exceptions::NotAcceptedException,
          exceptions::TransactionFailedException,
          exceptions::AlreadyExistsException
        );
```

```
cmiOrder::OrderStruct acceptOrderByProductNameNoAckV7(
   in cmiProduct::ProductNameStruct product,
   in cmiOrder::OrderEntryStruct anOrder)
      raises(
         exceptions::SystemException,
         exceptions::CommunicationException,
         exceptions::AuthorizationException,
         exceptions::DataValidationException,
         exceptions::NotAcceptedException,
         exceptions::TransactionFailedException,
         exceptions::AlreadyExistsException
                  );

cmiOrder::OrderStruct acceptOrderCancelReplaceRequestNoAckV7 (
   in cmiOrder::CancelRequestStruct cancelRequest,
   in cmiOrder::OrderEntryStruct newOrder)
      raises(
         exceptions::SystemException,
         exceptions::CommunicationException,
         exceptions::AuthorizationException,
         exceptions::DataValidationException,
         exceptions::NotAcceptedException,
         exceptions::TransactionFailedException
      );

cmiOrder::OrderStruct acceptStrategyOrderNoAckV7 (
   in cmiOrder::OrderEntryStruct anOrder,
   in cmiOrder::LegOrderEntryStructV2Sequence legEntryDetailsV2)
      raises(
         exceptions::SystemException,
         exceptions::CommunicationException,
         exceptions::AuthorizationException,
         exceptions::DataValidationException,
         exceptions::NotAcceptedException,
         exceptions::TransactionFailedException,
         exceptions::AlreadyExistsException
```

```
        );

    cmiOrder::OrderIdStruct acceptStrategyOrderV7(
      in cmiOrder::OrderEntryStruct anOrder,
      in cmiOrder::LegOrderEntryStructV2Sequence legEntryDetailsV2)
        raises(
          exceptions::SystemException,
          exceptions::CommunicationException,
          exceptions::AuthorizationException,
          exceptions::DataValidationException,
          exceptions::NotAcceptedException,
          exceptions::TransactionFailedException,
          exceptions::AlreadyExistsException
        );

    cmiOrder::OrderStruct acceptStrategyOrderCancelReplaceRequestNoAckV7(
      in cmiOrder::CancelRequestStruct cancelRequest,
      in cmiOrder::OrderEntryStruct newOrder,
      in cmiOrder::LegOrderEntryStructV2Sequence legEntryDetailsV2)
        raises(
          exceptions::SystemException,
          exceptions::CommunicationException,
          exceptions::AuthorizationException,
          exceptions::DataValidationException,
          exceptions::NotAcceptedException,
          exceptions::TransactionFailedException
        );

    cmiOrder::OrderIdStruct acceptStrategyOrderCancelReplaceRequestV7(
      in cmiOrder::CancelRequestStruct cancelRequest,
      in cmiOrder::OrderEntryStruct newOrder,
      in cmiOrder::LegOrderEntryStructV2Sequence legEntryDetailsV2)
        raises(
          exceptions::SystemException,
          exceptions::CommunicationException,
          exceptions::AuthorizationException,
```

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

cmiOrder::InternalizationOrderResultStructV2 acceptInternalizationOrderNoAckV7 (
  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
    );

cmiOrder::InternalizationOrderResultStructV2 acceptInternalizationStrategyOrderNoAckV7 (
  in cmiOrder::OrderEntryStruct primaryOrder,
  in cmiOrder::LegOrderEntryStructV2Sequence primaryOrderLegEntriesV2,
  in cmiOrder::OrderEntryStruct matchOrder,
  in cmiOrder::LegOrderEntryStructV2Sequence matchOrderLegEntriesV2,
  in cmiOrder::MatchType matchType)
    raises(
        exceptions::SystemException,
        exceptions::CommunicationException,
        exceptions::AuthorizationException,
        exceptions::DataValidationException,
        exceptions::NotAcceptedException,
        exceptions::TransactionFailedException
    );

cmiOrder::InternalizationOrderResultStruct acceptInternalizationStrategyOrderV7(
  in cmiOrder::OrderEntryStruct primaryOrder,
  in cmiOrder::LegOrderEntryStructV2Sequence primaryOrderLegEntriesV2,
```

```
        in cmiOrder::OrderEntryStruct matchOrder,
        in cmiOrder::LegOrderEntryStructV2Sequence matchOrderLegEntriesV2,
        in cmiOrder::MatchType matchType)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException,
                exceptions::NotAcceptedException,
                exceptions::TransactionFailedException
                            );

      cmiOrder::CrossOrderStruct acceptCrossingOrderNoAckV7 (
        in cmiOrder::OrderEntryStruct buyCrossingOrder,
        in cmiOrder::OrderEntryStruct sellCrossingOrder)
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException,
                exceptions::NotAcceptedException,
                exceptions::TransactionFailedException,
                exceptions::AlreadyExistsException
            );
  };

  interface Quote: cmiV5::Quote
  {
     cmiQuote::ClassQuoteResultStructV3Sequence acceptQuotesForClassV7(
        in cmiProduct::ClassKey classKey,
        in cmiQuote::QuoteEntryStructV4Sequence quotes )
            raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::DataValidationException,
```

```
            exceptions::NotAcceptedException,
            exceptions::TransactionFailedException
        );
    void acceptQuoteV7(
      in cmiQuote::QuoteEntryStructV4 quote )
        raises(
            exceptions::SystemException,
            exceptions::CommunicationException,
            exceptions::AuthorizationException,
            exceptions::DataValidationException,
             exceptions::NotAcceptedException,
            exceptions::TransactionFailedException
        );
};

interface UserSessionManagerV7 : cmiV6::UserSessionManagerV6
  {
    cmiV7::OrderEntry getOrderEntryV7()
      raises(
       exceptions::SystemException,
       exceptions::CommunicationException,
       exceptions::AuthorizationException
      );

    cmiV7::Quote getQuoteV7()
       raises(
          exceptions::SystemException,
          exceptions::CommunicationException,
          exceptions::AuthorizationException
       );

};

interface UserAccessV7
  {
    UserSessionManagerV7 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 cmiV8
{
        //-------------------------------------------------------------------------------------------------
        //      Trading Class  Query Service -
        //              Reports the trading class alvailability during trading hours, due to any
outage.
        //      This service is based on a subscription mechanism, once subscribed, users will be
notified
        //      of the unavailability of a Group or clases to trade for example a network outage,
        //      subsequently the user will be notified of the availability of a group or class to trade
once
        //      the problem is resolved during trading hours.
        //-------------------------------------------------------------------------------------------------

        interface TradingClassStatusQuery
        {
                cmiProduct::ProductGroupSequence getProductGroups()
                raises(
                        exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::DataValidationException,
                exceptions::NotFoundException,
```

```
            exceptions::AuthorizationException
                    );

            cmiProduct::ClassKeySequence getClassesForProductGroup(in
cmiProduct::ProductGroup productGroupName)
                    raises(
                            exceptions::SystemException,
                            exceptions::CommunicationException,
                            exceptions::DataValidationException,
                    exceptions::NotFoundException,
                            exceptions::AuthorizationException
                    );

            void subscribeTradingClassStatusForProductGroup(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ProductGroupSequence productGroupNames,
        in cmiCallbackV5::CMITradingClassStatusQueryConsumer clientListener)
                    raises(
                    exceptions::SystemException,
                    exceptions::CommunicationException,
                    exceptions::AuthorizationException,
                    exceptions::DataValidationException
                    );

            void subscribeTradingClassStatusForClasses(
        in cmiSession::TradingSessionName sessionName,
        in cmiProduct::ClassKeySequence classKeys,
        in cmiCallbackV5::CMITradingClassStatusQueryConsumer clientListener)
                    raises(
                    exceptions::SystemException,
                    exceptions::CommunicationException,
                    exceptions::AuthorizationException,
                    exceptions::DataValidationException
                    );
        };
```

```
        interface UserSessionManagerV8 : cmiV7::UserSessionManagerV7
        {
                TradingClassStatusQuery getTradingClassStatusQuery()
        raises(
                exceptions::SystemException,
                exceptions::CommunicationException,
                exceptions::AuthorizationException,
                exceptions::AuthenticationException,
                exceptions::NotFoundException
                );

        };

        interface UserAccessV8
        {
        UserSessionManagerV8 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 cmiCallbackV5
{
        interface CMITradingClassStatusQueryConsumer {
                void acceptTradingClassStatusUpdateforProductGroups(
```

```
        in cmiProduct::ProductGroupSequence listOfProductGroups,
        in cmiUtil::TradingClassStatusIndicator status);

                void acceptTradingClassStatusUpdateforClasses(
        in cmiProduct::ClassKeySequence listOfClasses,
        in cmiUtil::TradingClassStatusIndicator status);
        };
};

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;
    };
};
```

```
module cmiConstants

    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
const cmiOrder::ContingencyType INTERMARKET_SWEEP = 15; // Intermarket sweep (ISO)
const cmiOrder::ContingencyType RESERVE = 16; // Reserve order
const cmiOrder::ContingencyType MIDPOINT_CROSS    = 17; // Mid Point Cross
const cmiOrder::ContingencyType CROSS             = 18; // Cross
const cmiOrder::ContingencyType TIED_CROSS        = 19; // Tied cross
const cmiOrder::ContingencyType AUTOLINK_CROSS    = 20; // Auto link cross
const cmiOrder::ContingencyType AUTOLINK_CROSS_MATCH = 21; // Auto link cross
const cmiOrder::ContingencyType CROSS_WITHIN      = 22;
const cmiOrder::ContingencyType TIED_CROSS_WITHIN = 23;
const cmiOrder::ContingencyType STOCK_ODD_LOT_NBBO_ONLY = 24;
const cmiOrder::ContingencyType NBBO_FLASH_THEN_CANCEL = 25;
const cmiOrder::ContingencyType DO_NOT_ROUTE = 26;
const cmiOrder::ContingencyType NBBO_FLASH_RESPONSE = 27;
const cmiOrder::ContingencyType INTERMARKET_SWEEP_BOOK = 28; // Intermarket Sweep Book (ISB)
const cmiOrder::ContingencyType BID_PEG_CROSS = 29;
const cmiOrder::ContingencyType OFFER_PEG_CROSS = 30;
const cmiOrder::ContingencyType TIED_CROSS_SWEEP = 31;
const cmiOrder::ContingencyType CASH_CROSS = 32; // cash settlement cross
const cmiOrder::ContingencyType NEXT_DAY_CROSS = 33; // Next day settlement cross
const cmiOrder::ContingencyType TWO_DAY_CROSS  = 34; // Two day settlement cross
```

```
        };

interface TradingClassStatusIndicators
        {
                const short CLOSED_OUTAGE  = 1;
        const short OPEN_AFTER_OUTAGE     = 4;
        };

module cmiQuote
{
   typedef short RFQType;
   typedef long QuoteKey;
   typedef sequence <QuoteKey> QuoteKeySequence;
   typedef short QuoteUpdateControl;

   struct QuoteEntryStruct
   {
      cmiProduct::ProductKey productKey;
      cmiSession::TradingSessionName sessionName;
      cmiUtil::PriceStruct bidPrice;
      long bidQuantity;
      cmiUtil::PriceStruct askPrice;
      long askQuantity;
      string userAssignedId;
   };
   typedef sequence <QuoteEntryStruct> QuoteEntryStructSequence;

   struct QuoteEntryStructV3
   {
      cmiQuote::QuoteEntryStruct quoteEntry;
      cmiQuote::QuoteUpdateControl quoteUpdateControlId;
   };
   typedef sequence <QuoteEntryStructV3> QuoteEntryStructV3Sequence;

   struct QuoteEntryStructV4
   {
```

```
        cmiQuote::QuoteEntryStructV3 quoteEntryV3;
        cmiUtil::Side sellShortIndicator;
            string extensions;
    };
    typedef sequence <QuoteEntryStructV4> QuoteEntryStructV4Sequence;

#pragma use_factory_for_struct ON
    struct QuoteStruct
    {
        cmiQuote::QuoteKey quoteKey;
        cmiProduct::ProductKey productKey;
        cmiSession::TradingSessionName sessionName;
        string userId;
        cmiUtil::PriceStruct bidPrice;
        long bidQuantity;
        cmiUtil::PriceStruct askPrice;
        long askQuantity;
        long transactionSequenceNumber;
        string userAssignedId;
    };

    typedef sequence <QuoteStruct> QuoteStructSequence;

    struct QuoteStructV3
    {
        cmiQuote::QuoteStruct quote;
        cmiQuote::QuoteUpdateControl quoteUpdateControlId;
    };

#pragma use_factory_for_struct OFF

#pragma use_array_factory_for_1_dimension ON
typedef sequence <QuoteStructV3> QuoteStructV3Sequence;
#pragma use_array_factory_for_1_dimension OFF

    struct QuoteStructV4
```

```
{
   cmiQuote::QuoteStructV3 quoteV3;
   cmiUtil::Side sellShortIndicator;
   string extensions;
};

typedef sequence <QuoteStructV4> QuoteStructV4Sequence;

struct QuoteDetailStruct
{
   cmiProduct::ProductKeysStruct productKeys;
   cmiProduct::ProductNameStruct productName;
   cmiUtil::UpdateStatusReason statusChange;
   cmiQuote::QuoteStruct quote;
};
typedef sequence <QuoteDetailStruct> QuoteDetailStructSequence;

struct RFQEntryStruct
{
   cmiProduct::ProductKey productKey;
   cmiSession::TradingSessionName sessionName;
   long quantity;
};

struct RFQStruct
{
   cmiProduct::ProductKeysStruct productKeys;
   cmiSession::TradingSessionName sessionName;
   long quantity;
   long timeToLive;
   cmiQuote::RFQType rfqType;
   cmiUtil::TimeStruct entryTime;
};
typedef sequence <RFQStruct> RFQStructSequence;

struct QuoteFilledReportStruct
```

```
    {
        cmiQuote::QuoteKey quoteKey;
        cmiProduct::ProductKeysStruct productKeys;
        cmiProduct::ProductNameStruct productName;
        cmiOrder::FilledReportStructSequence filledReport;
        cmiUtil::UpdateStatusReason statusChange;
    };
    typedef sequence <QuoteFilledReportStruct> QuoteFilledReportStructSequence;

    struct ClassQuoteResultStruct
    {
        cmiProduct::ProductKey productKey;
        exceptions::ErrorCode errorCode;
    };
    typedef sequence <ClassQuoteResultStruct> ClassQuoteResultStructSequence;

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

    struct ClassQuoteResultStructV3
    {
        cmiQuote::ClassQuoteResultStructV2 quoteResult;
        cmiQuote::QuoteUpdateControl quoteUpdateControlId;
    };
    typedef sequence <ClassQuoteResultStructV3> ClassQuoteResultStructV3Sequence;
#pragma use_factory_for_struct OFF

    struct ClassQuoteResultStructV4
    {
        cmiQuote::ClassQuoteResultStructV3 quoteResultV3;
```

```
        cmiUtil::Side sellShortIndicator;
    string extensions;
};
typedef sequence <ClassQuoteResultStructV4> ClassQuoteResultStructV4Sequence;

struct QuoteRiskManagementProfileStruct
{
    cmiProduct::ClassKey classKey;
    long volumeThreshold;
    long timeWindow;
    boolean quoteRiskManagementEnabled;
};
typedef sequence <QuoteRiskManagementProfileStruct> QuoteRiskManagementProfileStructSequence;

struct UserQuoteRiskManagementProfileStruct
{
    boolean globalQuoteRiskManagementEnabled;
    cmiQuote::QuoteRiskManagementProfileStruct defaultQuoteRiskProfile;
    cmiQuote::QuoteRiskManagementProfileStructSequence quoteRiskProfiles;
};
struct QuoteBustReportStruct
{
    cmiQuote::QuoteKey quoteKey;
    cmiProduct::ProductKeysStruct productKeys;
    cmiProduct::ProductNameStruct productName;
    cmiOrder::BustReportStructSequence bustedReport;
    cmiUtil::UpdateStatusReason statusChange;
};
typedef sequence <QuoteBustReportStruct> QuoteBustReportStructSequence;

struct QuoteCancelReportStruct
{
    cmiQuote::QuoteKey quoteKey;
    cmiProduct::ProductKeysStruct productKeys;
    cmiProduct::ProductNameStruct productName;
    cmiUtil::ActivityReason cancelReason;
```

```
    cmiUtil::UpdateStatusReason statusChange;
  };
  typedef sequence <QuoteCancelReportStruct> QuoteCancelReportStructSequence;

  struct QuoteDeleteReportStruct {
    cmiQuote::QuoteDetailStruct quote;
    cmiUtil::ActivityReason deleteReason;
  };
  typedef sequence <QuoteDeleteReportStruct> QuoteDeleteReportStructSequence;

  struct LockNotificationStruct {
    cmiSession::TradingSessionName sessionName;
    cmiProduct::ProductType productType;
    cmiProduct::ClassKey classKey;
    cmiProduct::ProductKey productKey;
    cmiUtil::Side side;
    cmiUtil::PriceStruct price;
    long quantity;
    string extensions;
    cmiUser::ExchangeAcronymStructSequence buySideUserAcronyms;
    cmiUser::ExchangeAcronymStructSequence sellSideUserAcronyms;
  };
  typedef sequence <LockNotificationStruct> LockNotificationStructSequence;
```

```
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;
   typedef long long Identifier;
   typedef short LinkageIndicatorReturnType;
   typedef string RecapSuffix;
   typedef short TradingClassStatusIndicator;
```

```
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 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;
```

Document Changes

API-01

- No changes

API-02

- Incorporated new sections to include the CMi V7 and CMI V8 functionality.
- Updated the contingency mapping table to include:

```
const cmiOrder::ContingencyType CASH_CROSS = 32;
const cmiOrder::ContingencyType NEXT_DAY_CROSS = 33;
const cmiOrder::ContingencyType TWO_DAY_CROSS  = 34;
```

API-03

- Added values for the new constants based on this release.
- Provided new class diagrams for CMi V7 interfaces.

API-04

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

API-05

- No changes

API-06

- Updated the Connection Method section to remove the text, “dial up via PPP or ISDN connections.” Replaced the text with: “dial is accomplished via VPN as described in the NET-01 document.”
- Changed the ORB CAS port number from 8100 to 8102.
- Added CBOE approved Extranets as a second connection method for the production environment.
- Removed terms that no longer apply from the Glossary section.

API-07

- No changes

API-08

- No changes

CAS-01

- No changes

CAS-02

- No changes

Simulator

- CMi simulator updates to include functionality for CMi V7 interfaces.

Test Plan Changes

- No changes