Application Programming Interface

Version 9.0.2

CBOE API Volume 8: CMi Programmer's Guide to the Market Data Express (MDX) Data Feed

Programmer's Guide to Interfaces and Operations for CBOE's Market Interface (CMI) for MDX

CBOE PROPRIETARY INFORMATION

15 July 2011

Document #API-08

Front Matter

Disclaimer

Table of Contents

Change Notices

The following change notices are provided to assist users of the CMi in determining the impact of changes to their applications.

Date	**Version**	**Description of Change**
15 Jul 2011	V9.0.2	No changes
29 Apr 2011	V9.0.1	No changes
14 Jan 2011	V9.0	No changes
08 Jan 2010	V7.0	No changes
21 Nov 2008	V5.3	No changes
03 Oct 2008	V5.2	No changes
23 Jul 2008	V5.1	No changes
29 Feb 2008	V5.0	No changes
18 Jan 2008	V4.2.4	No changes
02 Nov 2007	V4.2.3	No changes
13 Jun 2007	V4.2.2	Updated the Oneway CORBA Calls section in relation to sequence numbers
01 June 2007	V4.2.2	No changes
23 Feb 2007	V4.2.1	No changes
15 Dec 2006	V4.2	No changes
20 Sept 2006	V4.1	No changes
01 Jun 2005	V4.0	Updated the LastSaleStructV4: netPriceChange description
		Added definitions for Sale Conditions
25 May 2006	V4.0	New document for CBOE MDX

About This Document

Purpose

This document provides details on how to create an application to access the CBOE MDX Data Feed via the CBOE Market Interface (CMi).

Intended Audience

Software developers using the CMi to develop applications that use the CBOE MDX Data Feed.

Prerequisites

This document assumes that the reader has a working knowledge of CORBA and one of the programming languages supported by CORBA, such as C++ or Java. See the reference section of a list of books and web sites that can provide you with fundamentals on CORBA. Specifically, you should be familiar with CORBA modules, interfaces, structs, operations, IORs, exceptions, and callbacks.

Support and Questions Regarding the CBOE APIs

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.

Introduction

The CBOE is adding a new interface to support the MDX data feed. This interface is designed to generically support the reporting of quotes, trades, and other information for options, stocks, and futures. The interface is an Application Programming Interface (API) that provides access to market data and is targeted at firms who desire another source of market data. This API is known as the *CBOE Market Interface* (CMi). CMi is a distributed object interface based upon the CORBA (Common Object Request Broker Architecture) standard from the Object Management Group (OMG). The interface is defined using the Interface Definition Language (IDL), which is an OMG and ISO standard. Messages are transported using the Internet Inter-Orb Protocol (IIOP), which operates over standard Internet protocols (TCP/IP).

Introduction to the CBOE Market Interface (CMi)

The CMi is a set of CORBA Interfaces that expose services provided by the CBOE Application Server. Care has been taken to use CORBA as a transport layer, as opposed to a naïve approach of partitioning an object-oriented application's objects across multiple processes and networks. Services are exposed and messages are passed between a client application and the CBOE Application Server (CAS). The messages are in the form of CORBA structs and sequences of CORBA structs.

Getting Started

Choose Your Development Environment

CBOE has adopted the use of CORBA IDL as an interface to exchange services because there are a myriad of alternatives CORBA provides to users, in terms of development environment. Users of CMi are able to select the language that best suits their own needs, they can choose the development and deployment platforms that best fit within their environment, and they are able to use CORBA middleware that best fits their internal needs.

CBOE recommends that you use one of the following languages:

- C++
- Java

Select a Development Platform:

- Microsoft Windows NT 4.0 Service Pack 6 or later
- Sun Solaris Version 2.6 or later

Select a CORBA middleware product:

- Java 1.4 JDK from Sun Microsystems (only for testing, not production)
- omniORB from AT&T Laboratories Cambridge

- Tao Orb from Object Computing or Washington University
- JacORB 2.2.3 from www.jacorb.org.

Acquire the CMi Software

Retrieve the CMi Software Developer’s Kit from the CBOE Internet Web site http://systems.cboe.com. This site requires you to contact CBOE to obtain a User ID and password. The CMi is available in zip and tar formats. The CMi software includes:

1. CMi IDL
2. CAS Simulator
3. Documentation, including this programmer’s guide
4. Example programs in Java and C++
5. Scripts for supported development environments

Implement Your Application using the Software Developer’s Kit

Use the CBOE CMi Software Developer’s Kit to begin developing your application. This is done via the following steps:

1. Compile the CMi IDL provided with the CMi Software Developer’s Kit using the CORBA compiler for the middleware product that you have selected and procured separately.
2. Determine how to integrate the MDX data feed within your existing or planned application(s). Examples are provided.
3. Implement the mdx data feed using the interfaces and operations provided.
4. Use the CAS Simulator to test your application in a stand-alone testing mode.
5. Follow the procedures for certifying your application as outlined in API-01: Volume1: Overview and Concepts.

Using the CMi

The CMi is composed of a set of CORBA modules. There are four types of modules provided with CMi

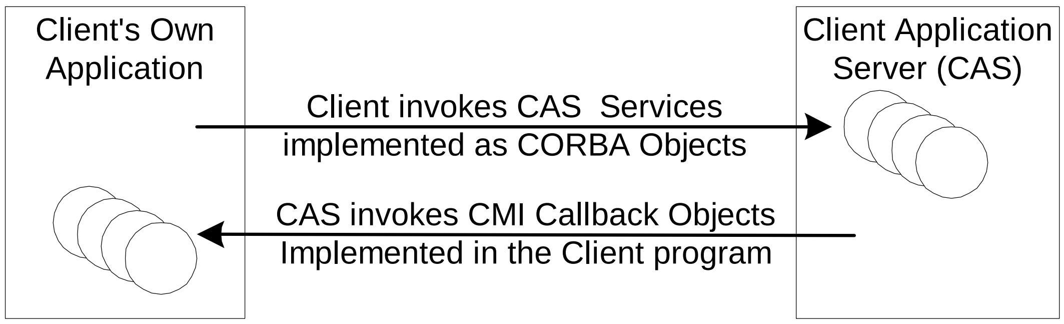
1. The interfaces and operations (or methods) provided or required by the CBOE Application Server (CAS).
2. CORBA structs that are the messages that are passed between the client and the CAS.
3. Modules containing constant values that are defined in the CMi.
4. Modules that contain interfaces for exceptions and error codes.

Definition: A CORBA Operation is a function or behavior that is provided by a distributed object. An operation is the CORBA terminology for an object oriented method.

It is best to start with the interfaces provided in the CMi to understand the functionality provided by the CAS through the CMi. The operations, provided by these interfaces, constitute the functionality of the CAS. The MDX-specific interfaces and operations are contained in the cmiV4.idl and cmiCallbackV4.idl files.

Interfaces

The programming model provided through the CMi interface supports access to services that are implemented as CORBA objects on the CAS and CORBA callback objects that you implement in your application program. These callback objects are registered with the CAS via operations available in the various CMi interfaces.


Client's Own Application
Client Application Server (CAS)
Client invokes CAS Services implemented as CORBA Objects
CAS invokes CMI Callback Objects Implemented in the Client program

Communication between the CAS and the client programs is done by passing messages. These messages are implemented as CORBA structs. The CORBA struct is very similar to a struct in C or C++. Anyone familiar with any higher level language (C, C++, COBOL) should be able to read and understand the structs (or messages) that are used with the CMi. The structs for a particular interface are placed in CORBA modules. The CORBA module name that contains the structs are named to correspond to the interfaces provided by the CMi. For instance the *MarketQuery interface* primarily operates on structs from the *cmiMarketData module*.

Operation Naming Conventions

A consistent naming convention has been used in naming operations. The following naming conventions have been used throughout the CAS.

Operation Prefix	Behavior
get* Example: cmi::ProductQuery::getProductClasses()	Query a CAS service for information. Information is returned as part of the call – often as a sequence of CORBA structs. Some get operations require the client to

Operation Prefix	Behavior
	provide a callback (or consumer) object to receive subsequent updates to the information that was requested.
accept* Example: cmiCallbackV4::CMITickerConsumer::acceptAcceptTicker()	Accept operations are used to deliver information to an interface. CAS Services have accept* operations to accept messages from the client. Client callbacks have operations to accept messages from the CAS.
Subscribe* Example: cmiV4::MarketQuery::subscribeTicker()	Subscribe operations permit the client to subscribe to information published by the CAS. The subscription operation always requires a callback object to be provided as an argument. The CAS will call an accept* operation on the callback object to provide the information that was subscribed for by the client. *NOTE: You must register your callback with the client ORB to receive information.*
Unsubscribe* Example: cmiV4::MarketQuery::unsubscribeTicker()	Unsubscribe operations permit the client application to unsubscribe for information that was previously subscribed for using either a get operation or a subscribe operation.

CMi Interfaces

The CAS provides many interfaces to the client. MDX users will be allowed to use the following services defined in the *cmi modules* (cmi.idl and cmiV4.idl).

Operation Prefix	Behavior
cmiV4::MarketQuery	The MarketQuery service provides subscription methods to the current market, ticker, recap and last sale.
cmi::ProductQuery	The Product Query service provides for the retrieval of product and class information.
cmiV4::UserAccessV4	The UserAccess Service provides the capability to logon to the CAS. This object is accessed across the network by making an HTTP Get query to the CAS to retrieve the object reference (as a stringified IOR). A successful logon to the CAS will result in an object reference to the client's UserSessionManager Service running on the CAS being returned.
cmiV4::UserSessionManagerV4	The User Session Manager Service provides

Operation Prefix	Behavior
	operations to return object references to the other services provided by the CAS. The UserSessionManager also returns information on the current logged in user and provides an operation to change the password for the currently logged in user. Reference to the User Session ManagerV4 can be obtained through login using the User AccessV4 interface.
cmi::Version	Returns the version of the IDL that was used to create the CAS.

The CAS objects listed above are made available via the following steps. The *UserAccess Interface* permits you to login. A successful login request via the *UserAccess Interface* results in the return of the reference to the *UserSessionManager Interface* for your application. You then use the *UserSessionManager* to access references to the remaining interfaces provided on the CAS.

Modules that Contain the Message Formats

The following table contains the list of CORBA IDL modules that contain the messages sent between CAS and client. These messages are implemented as CORBA structs.

Module	Description
cmiConstants	This interfaces describes all the constant values used in all the CMi messages.
cmiMarketData	The cmiMarketData module containts the CORBA structs that are sent to the client callback functions via their accept* methods. The four structs used in MDX are CurrentMarketStructV4, TickerStructV4, RecapStructV4 and LastSaleStructV4.
cmiProduct	Messages describing products within the system.
cmiUser	Contains the UserLoginStruct used for logging on.

MDX Application Design

A basic MDX application has several steps to follow to make use of the data feed:

1. Logon to the CAS. See the section: *Logging onto the CAS*.
2. Download class information/subscribe to class updates. Refer to the section: *Downloading Class Data* below.
3. Subscribe and unsubscribe types of data as necessary. Reference the section: *Subscribing to Class Data* below.

Client Application Access to the CAS

Logging onto the CAS

You must first logon to the CAS in order to obtain access services. The UserAccessV4 service provides a logon() operation. There are several steps that must be completed prior to invoking the logon() operation.

Step 1. Implement the CMIVersion object. This is used during login to determine what version of the CMi that's being used. See the examples com.cboe.examples.common.CMIVersion and CMIVersion.h.

Step 2. Implement a CMIUserSessionAdmin call back object and register it with your ORB so that it is available for use by the CAS. The CMIUserSessionAdmin object has an operation for accepting a heartbeat from the CAS. See the example com.cboe.examples.common.UserSessionAdminCallback and UserSessionAdminCallback.cpp. Also, see the section: *Heartbeat Callback* below.

Step 3. You must create and populate a UserLogonStruct, which includes the userid, password, a CMI version, and session mode. Use the CMIVersion object you created in Step 1. The session mode should be LoginSessionTypes::PRIMARY.

Step 4. You must obtain the object reference to the UserAccessV4 object located on the CAS. CBOE has provided source code for a UserAccessLocator class that contacts the CAS using the http protocol. For exact details, refer to the section below: *Obtaining the User Access Initial Object Reference*.

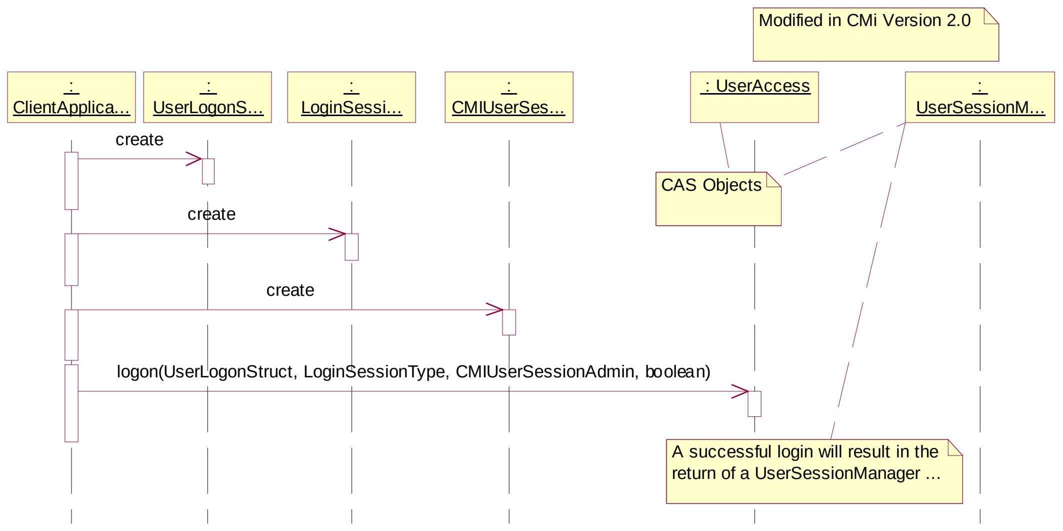
Step 5. Invoke the UserAccessV4.logon() operation, passing as arguments the UserLogonStruct and the CMIUserSessionAdmin object.

Step 6. To logoff cleanly use the UserAccessV4.logoff() method.

The example programs from CBOE use a single class that handles all the CAS interactions: CASAcessManager (see com.cboe.examples.common.CASAccessManager and CASAccessManager.cpp). Through the CASAccessManager you can logon to the CAS and access all the CAS Services on an as needed basis. Refer to the section: *A CAS Access Manager*.

The following is a sequence diagram depicting the CAS logon process.

CAS Logon Process


Modified in CMi Version 2.0
: ClientApplica...
: UserLogonS...
: LoginSessi...
: CMIUserSes...
: UserAccess
: UserSessionM...
create
CAS Objects
create
create
logon(UserLogonStruct, LoginSessionType, CMIUserSessionAdmin, boolean)
A successful login will result in the return of a UserSessionManager ...

Heartbeat Callback

CBOE sends the firm's application a heartbeat every 2 seconds. The Orb on which the CAS is now based uses a timeout for heartbeat. Currently this timeout is set to be 20 seconds. This means when a CMi client application doesn't respond to a single heartbeat request within 20 seconds for any reason, the connection will be considered dead and the CAS will logoff the connection and clean up its session. As a result, care must be taken to make sure that the heartbeat thread in your application never becomes starved or isn't serviced or your application may be logged off on a false detection.

Obtaining the User Access Initial Object Reference

UserAccessLocator is used to obtain the Interoperable Object Reference (IOR) for the UserAccessV4 interface. CORBA objects are referenced by their Interoperable Object Reference. The CAS provides an IOR to access the UserAccessV4 object via an http connection. This is commonly referred to as the bootstrap process: the client application obtains an initial IOR via some out of band protocol, in this case HTTP. The TCP port number for the http server in the CAS is set to a default value of 8001. The TCP port number can be changed via a CAS configuration parameter if need be to avoid potential conflicts with other software that may be running with the CAS.

Each client application program that wants to access the CAS must first obtain the IOR for the UserAccessV4 service. Once this is obtained the Client Application Program must narrow the reference and then bind to the UserAccess object in the CAS. See com.cboe.examples.common.UserAccessLocator or UserAccessLocator.h. The IOR is by standard a variable length message that can differ in implementation from ORB to ORB. This IOR will normally be split across multiple packets. Use the supplied length field in the IOR string.

Source Code for UserAccessLocator

```
package com.cboe.examples.common;

import java.net.*;
import java.io.*;
import java.util.*;

/**
 *
 * Access to the CAS is accomplished by acquiring an interoperable
 * object reference to the UserAccess object on the CAS using the
 * http protocol. This class communicates to a CAS - specified by
 * its IP address and and TCP Port number.
 * <br><br>
 * Copyright © 1999-2001 by the Chicago Board Options Exchange ("CBOE"), as an unpublished
work.
 * The information contained in this software program constitutes confidential and/or trade
 * secret information belonging to CBOE. This software program is made available to
 * CBOE members and member firms to enable them to develop software applications using
 * the CBOE Market Interface (CMi), and its use is subject to the terms and conditions
 * of a Software License Agreement that governs its use. This document is provided "AS IS"
 * with all faults and without warranty of any kind, either express or implied.
 *
 * @version 1.0
 *
 * @author Jim Northey
 *
 * @since Version 0.5
 */

public class UserAccessLocator {

  String ipAddress;
  int tcpPortNumber;

  final String IOR_REFERENCENAME = "/UserAccess.ior";
  final String DEFAULT_CAS_IP = "localhost";
  final int DEFAULT_CAS_TCP_PORT = 8003;

  String httpResponse; // Contains the HTTP Header information returned
                       // by the Server

  /**
   *  Construct a UserAccessLocator Object
   *
   * @param String ipAddress The IP Address for the CAS
   * @param String tcpPortNumber The Port Number of the HTTP Server on the CAS
   *
```

```
    */
   public UserAccessLocator(String ipAddress, int tcpPortNumber) {
     this.ipAddress = ipAddress;
     this.tcpPortNumber = tcpPortNumber;

   }
   /**
    *  Construct a UserAccessLocator Object using a default
    *  IP Address of DEFAULT_CASIP and TCP Port number of DEFAULT_CASTPCPORT.
    *
    */
   public UserAccessLocator() {
          this.ipAddress = DEFAULT_CAS_IP;
          this.tcpPortNumber = DEFAULT_CAS_TCP_PORT;
   }
   /**
    * Obtain the IOR for the UserAccess object of the CAS using the HTTP protocol
    *
    */
   public String obtainIOR() throws com.cboe.exceptions.CommunicationException {

       String ior;

       int numFields = 0;

       URLConnection conn;

       StringBuffer httpRespBfr = new StringBuffer();

       try {
         URL url = new URL("http",ipAddress,tcpPortNumber,IOR_REFERENCENAME);

         conn = url.openConnection();

         //
         // Skip over the headers in the return message
         // Save the headers in the event the user wants
         // to access them
         //

         String s = null;
         for(numFields=0; ; numFields++) {
             s = conn.getHeaderField(numFields);
             if (s == null) break;
             httpRespBfr.append(s);
             httpRespBfr.append("\n");
         }

         httpResponse = httpRespBfr.toString();

         if (numFields == 0) {
            com.cboe.exceptions.CommunicationException e = new
com.cboe.exceptions.CommunicationException();
            e.details = new com.cboe.exceptions.ExceptionDetails();
            e.details.message = "No CAS Found at IP Address: "+ipAddress+ " Port:
"+tcpPortNumber;
            e.details.dateTime =
DateTimeHelper.dateTimeStructToString(DateTimeHelper.makeDateTimeStruct(new Date()));
            e.details.error = 9000;
            e.details.severity = 1;
            throw e;
         }

         BufferedReader in  = new BufferedReader(
               new InputStreamReader((InputStream)conn.getContent()));

         ior = in.readLine();

       }
```

```
      catch(java.io.IOException e) {
           com.cboe.exceptions.CommunicationException cmiexception = new
com.cboe.exceptions.CommunicationException();
           cmiexception.details = new com.cboe.exceptions.ExceptionDetails();
           cmiexception.details.message = e.getMessage();
           cmiexception.details.dateTime =
DateTimeHelper.dateTimeStructToString(DateTimeHelper.makeDateTimeStruct(new Date()));
           cmiexception.details.error = 9000;
           cmiexception.details.severity = 1;
           throw cmiexception;
      }


      return ior;
  }
  /**
   * Return the HTTP Response that was obtained from the CAS Server
   *
   */
  public String getHttpResponse() {
        return httpResponse;
  }

  /**
   * Provide a way for the user to query what IP address was used to access the CAS
   */
  public String getCASIPAddress() {
        return ipAddress;
  }
  /**
   * Provie a way for the user to query what TCP Port number was used to access the CAS
   */
  public int getTCPPortNumber() {
        return tcpPortNumber;
  }
}
```

Accessing CAS Services

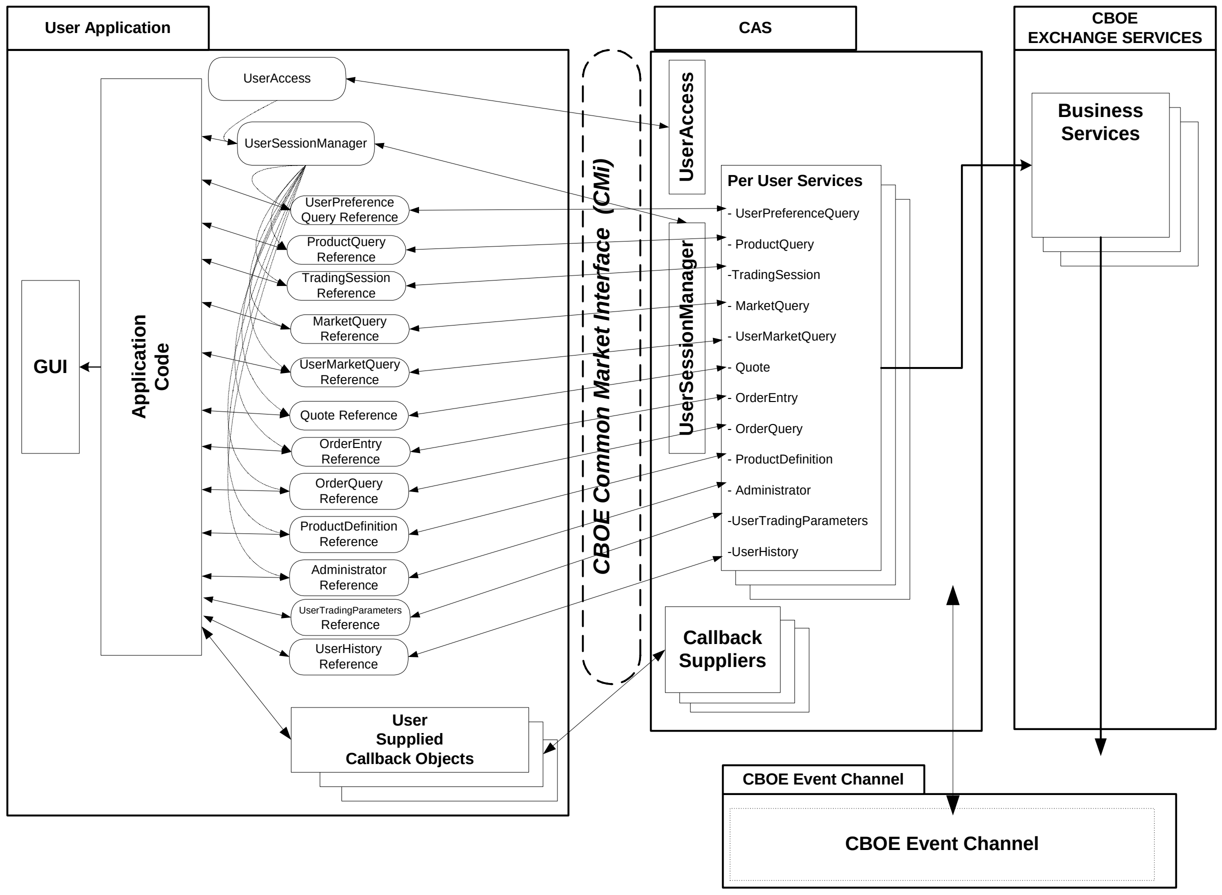
Once the object reference to the UserSessionManagerV4 has been returned from the UserAccessV4::logon() operation, the client application can obtain access to CAS services by invoking operations provided by the UserSessionManagerV4 to each CAS service. Here is a summary of CAS services relevant to MDX.

Note: this is a subset of the full set of CAS services.

Summary of UserSessionManager V4 Operations

UserSessionManagerV4 Operations	**Description**
authenticate	Used to re-authenticate the client to the CAS after a period of inactivity is detected by the CAS. This method is invoked as a response to an invocation of The UserSessionAdminCallback.acceptAuthenticationNotice() operation. This method is required.
changePassword	This operation is provided to change the password for the user currently logged into the CAS. You must specify the old password and the new password. Developers of interactive applications are strongly encouraged to prompt the user to confirm the new password beforehand and then validate the new password by comparing the new password with the confirmed version prior to invoking the changePassword() operation. This method is required in case the CBOE requires users to change their password at some point in time for security reasons. This method is required.
getMarketQuery	Obtain the reference to the Market Query Service. This method is required.
getProductQuery	Obtain the reference to the Product Query Service. This method is required.
getSystemDateTime	Returns the System Date and Time from the CAS. This method is optional.
getVersion	Obtain the reference to the Version Service. A VersionLabel is returned from the CAS. It is crucial to match the version of CMi that the client is using with the version that the CBOE is using. This method is required.
Logout	Logs the current user off of the CAS. This method is required.

The following diagram shows the architecture of a typical application and the sequence of steps in obtaining references to CAS provided services.


User Application
UserAccess
UserSessionManager
GUI
Application Code
UserPreference Query Reference
ProductQuery Reference
TradingSession Reference
MarketQuery Reference
UserMarketQuery Reference
Quote Reference
OrderEntry Reference
OrderQuery Reference
ProductDefinition Reference
Administrator Reference
UserTradingParameters Reference
UserHistory Reference
User Supplied Callback Objects
CBOE Common Market Interface (CMi)
CAS
UserAccess
UserSessionManager
Per User Services
- UserPreferenceQuery
- ProductQuery
-TradingSession
- MarketQuery
- UserMarketQuery
- Quote
- OrderEntry
- OrderQuery
- ProductDefinition
- Administrator
-UserTradingParameters
-UserHistory
Callback Suppliers
CBOE EXCHANGE SERVICES
Business Services
CBOE Event Channel
CBOE Event Channel

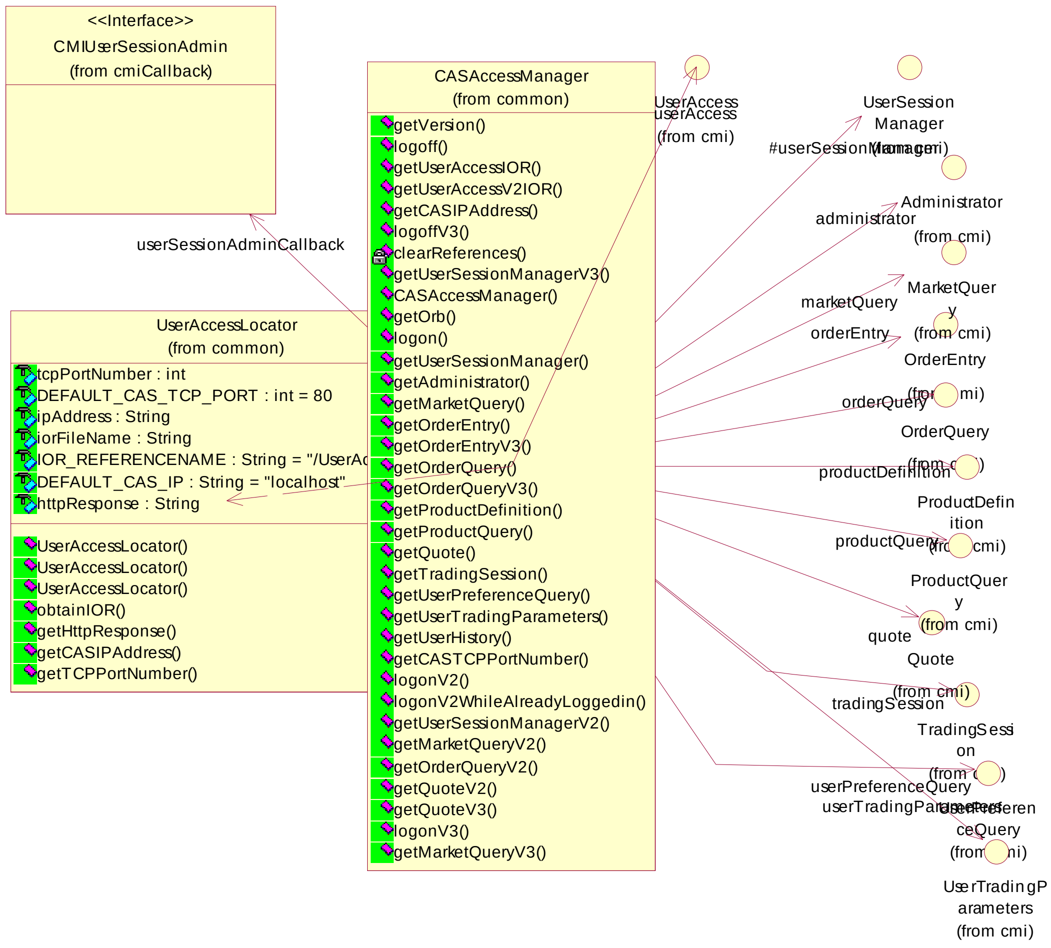
A CAS Access Manager

CBOE has created an example class that manages access to the CAS. It is provided as one possible approach for access to the CAS.

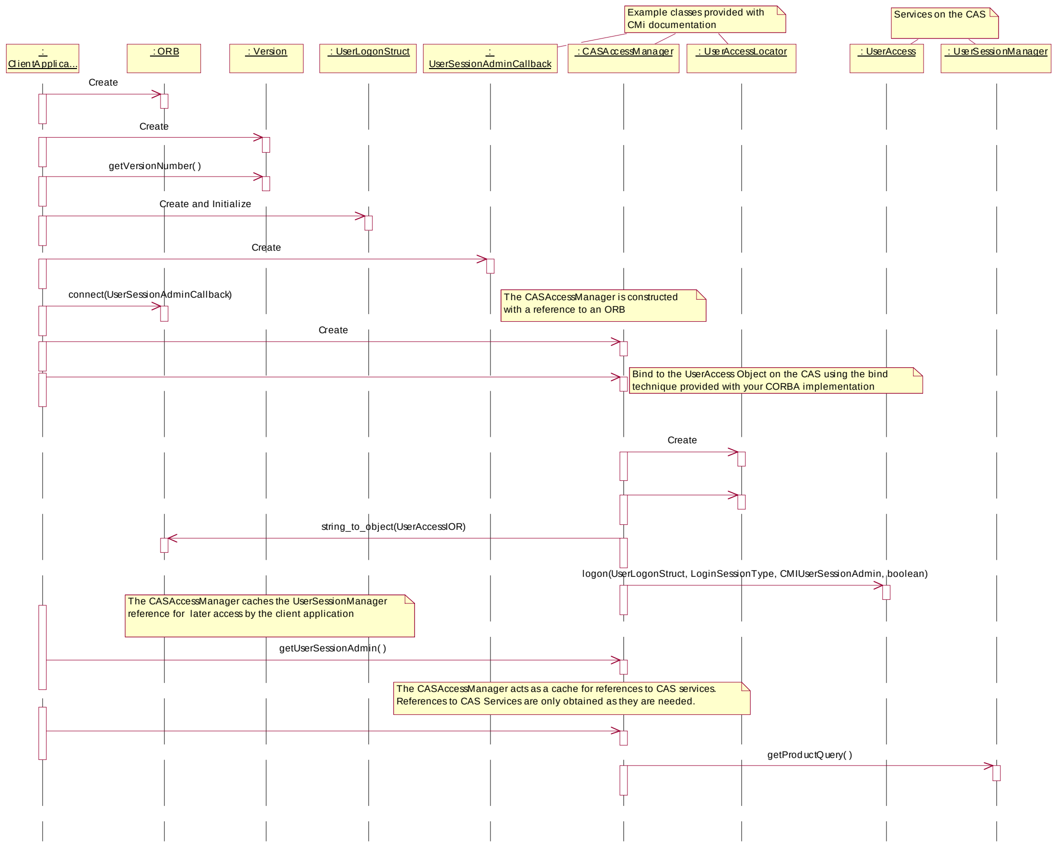
The com.cboe.examples.common.CASAccessManager (or CASAccessManager.cpp) encapsulates the behavior of obtaining the Interoperable Object Reference (IOR) from the CAS HTTP server, creating an object reference to a UserAccessV4 object, logging on to the UserAccessV4 object. The CASAccessManager then caches the object reference to the UserSessionManagerV4 service on the CAS. Methods to access other CAS services are provided. The CASAccessManager obtains the CAS object references on an as needed basis and caches them for subsequent usage by applications.

The Java example programs provided with the CAS all use this CASAccessManager class. A similar implementation is provided in C++ and is used in selected C++ examples.

CAS Access Manager Class Diagram



CAS Access Manager Sequence Diagram


Example classes provided with CMi documentation
Services on the CAS
: ClientApplica...
: ORB
: Version
: UserLogonStruct
: UserSessionAdminCallback
: CASAccessManager
: UserAccessLocator
: UserAccess
: UserSessionManager
Create
Create
getVersionNumber( )
Create and Initialize
Create
connect(UserSessionAdminCallback)
The CASAccessManager is constructed with a reference to an ORB
Create
Bind to the UserAccess Object on the CAS using the bind technique provided with your CORBA implementation
Create
string_to_object(UserAccessIOR)
logon(UserLogonStruct, LoginSessionType, CMIUserSessionAdmin, boolean)
The CASAccessManager caches the UserSessionManager reference for later access by the client application
getUserSessionAdmin( )
The CASAccessManager acts as a cache for references to CAS services. References to CAS Services are only obtained as they are needed.
getProductQuery( )

CAS Access Manager Source Code

```
package com.cboe.examples.common;

import java.util.Date;

import com.cboe.exceptions.AuthorizationException;
import com.cboe.exceptions.CommunicationException;
import com.cboe.exceptions.ExceptionDetails;
import com.cboe.exceptions.SystemException;
import com.cboe.idl.cmiConstants.LoginSessionTypes;
import com.cboe.idl.cmiV3.UserSessionManagerV3;
import com.cboe.idl.cmiV4.UserSessionManagerV4;

/**
 * Manages access to a CAS, including logging on to the CAS, returning object
 * references to CAS Interfaces, creation and registration of the
 * CMIUserSessionAdmin callback object. <br>
 * <br>
 * Copyright © 1999-2006 by the Chicago Board Options Exchange ("CBOE"), as an
 * unpublished work. The information contained in this software program
 * constitutes confidential and/or trade secret information belonging to CBOE.
 * This software program is made available to CBOE members and member firms to
 * enable them to develop software applications using the CBOE Market Interface
 * (CMi), and its use is subject to the terms and conditions of a Software
 * License Agreement that governs its use. This document is provided "AS IS"
 * with all faults and without warranty of any kind, either express or implied.
 *
 * @version 4.0
 *
 * @author JN
 * @author JZW
 *
 */

public class CASAccessManager {

        protected org.omg.CORBA.ORB orb; // Orb provided to CASAccessManager when
                                                                    // it was
constructed

        String userAccessIOR; // Stringified IOR returned by CAS

        String casVersion; // Version returned by CAS as a string

        static final boolean gmdTextMessaging = true;

        public static String USER_ACCESS_V2_REFERENCENAME = "/UserAccessV2.ior";

        public static String USER_ACCESS_V3_REFERENCENAME = "/UserAccessV3.ior";

        public static String USER_ACCESS_V4_REFERENCENAME = "/UserAccessV4.ior";

        String userAccessV2IOR;

        String userAccessV3IOR;

        String userAccessV4IOR;

        String casIPAddress; // Keep the CAS IP Address and TCP Port Number
                                                   // around

        int casTCPPortNumber; // in case the user wants to access it

        //
        // References to the CAS Interfaces
        //
        // The CASAccessManager is responsible for obtaining the references
        // to the CAS interfaces on an as needed basis.
        //
        com.cboe.idl.cmi.Administrator administrator;

        com.cboe.idl.cmi.UserAccess userAccess;

        protected com.cboe.idl.cmi.UserSessionManager userSessionManager;

        com.cboe.idl.cmi.OrderEntry orderEntry;

        com.cboe.idl.cmi.OrderQuery orderQuery;

        com.cboe.idl.cmi.TradingSession tradingSession;

        com.cboe.idl.cmi.ProductDefinition productDefinition;

        com.cboe.idl.cmi.MarketQuery marketQuery;

        com.cboe.idl.cmi.Quote quote;

        com.cboe.idl.cmi.ProductQuery productQuery;

        com.cboe.idl.cmi.UserPreferenceQuery userPreferenceQuery;

        com.cboe.idl.cmi.UserTradingParameters userTradingParameters;

        com.cboe.idl.cmi.UserHistory userHistory;

        // services defined in cmiV2
        com.cboe.idl.cmiV2.Quote quoteV2;

        com.cboe.idl.cmiV2.MarketQuery marketQueryV2;

        com.cboe.idl.cmiV2.OrderQuery orderQueryV2;

        com.cboe.idl.cmiV2.UserAccessV2 userAccessV2;

        com.cboe.idl.cmiV2.SessionManagerStructV2 sessionManagerStructV2;

        com.cboe.idl.cmiV2.UserSessionManagerV2 userSessionManagerV2;

        // service defined in cmiV3
        com.cboe.idl.cmiV3.OrderEntry orderEntryV3;

        com.cboe.idl.cmiV3.OrderQuery orderQueryV3;

        com.cboe.idl.cmiV3.Quote quoteV3;

        com.cboe.idl.cmiV3.MarketQuery marketQueryV3;

        com.cboe.idl.cmiV3.UserAccessV3 userAccessV3;

        com.cboe.idl.cmiV3.UserSessionManagerV3 userSessionManagerV3;

        // service defined in cmiV4
        com.cboe.idl.cmiV4.UserAccessV4 userAccessV4;

        com.cboe.idl.cmiV4.UserSessionManagerV4 userSessionManagerV4;

        com.cboe.idl.cmiV4.MarketQuery marketQueryV4;

        /**
         * The CASAccessManager is responsible for providing a CMIUserSessionAdmin
         * interface to the CAS
         */
        com.cboe.idl.cmiCallback.CMIUserSessionAdmin userSessionAdminCallback;
        /**
         * Constructs a CAS Access Manager
         *
         * @param orb
         *            ORB provided by client
         */
        public CASAccessManager(org.omg.CORBA.ORB orb) {
               this.orb = orb;
        }

        public org.omg.CORBA.ORB getOrb() {
               return this.orb;
        }

        /**
         * Logon to the CAS
         *
         * @param userLogonStruct
         * @param userSessionAdminCallback
         *            Provide a constructed userSession admin callback object. The
         *            client is responsible
         * @param casIPAddress
         *            The IP Address or domain name of the CAS
         * @param casTCPPortNumber
         *            The TCP Port Number of the CAS HTTP Server
         */
        public void logon(
                      com.cboe.idl.cmiUser.UserLogonStruct userLogonStruct,
                      com.cboe.idl.cmiCallback.CMIUserSessionAdmin
userSessionAdminCallback,
                      String casIPAddress, int casTCPPortNumber)
                      throws com.cboe.exceptions.SystemException,
                      com.cboe.exceptions.CommunicationException,
                      com.cboe.exceptions.AuthorizationException,
                      com.cboe.exceptions.AuthenticationException,
                      com.cboe.exceptions.DataValidationException {

               this.userSessionAdminCallback = userSessionAdminCallback;

               // orb.connect(userSessionAdminCallback); // We will connect the object
               // for the user

               this.casIPAddress = casIPAddress;
               this.casTCPPortNumber = casTCPPortNumber;

               UserAccessLocator locator = new UserAccessLocator(casIPAddress,
                              casTCPPortNumber);

               //
               // UserLocator.obtainIOR() will throw a
               // com.cboe.exceptions.CommunicationException
               // if it cannot talk to the CAS. This will be handled by the
               // CASAccessManager client.
               //
               this.userAccessIOR = locator.obtainIOR();

               org.omg.CORBA.Object objRef;

               objRef = orb.string_to_object(userAccessIOR);

               //
               // The UserAccessHelper Class is generated by the CORBA IDL
               // compiler for Java.
               //

               userAccess = com.cboe.idl.cmi.UserAccessHelper.narrow(objRef);

               short sessionType = LoginSessionTypes.PRIMARY;
               //
               // Exceptions thrown by the UserAccess.logon() operation will be handled
               // by the CASAccessManager client.
               //
               userSessionManager = userAccess.logon(userLogonStruct, sessionType,
                              userSessionAdminCallback, gmdTextMessaging);

               return;
        }

        /**
         * Return the object reference for the UserSessionManager interface on the
         * CAS
         *
         */
        public com.cboe.idl.cmi.UserSessionManager getUserSessionManager()
                      throws com.cboe.exceptions.AuthorizationException {

               if (userSessionManager == null) {
                      com.cboe.exceptions.AuthorizationException authorizationException =
new com.cboe.exceptions.AuthorizationException();
                      short severity = 1;
                      authorizationException.details = new
com.cboe.exceptions.ExceptionDetails(
                                     "CASAccessManager Error: Trying to access
UserSessionManager before being logged on or after being logged off",
                                     new Date().toString(), severity, 9999);
                      throw authorizationException;
               }

               return userSessionManager;
        }

        /**
         * Return the object reference for the Administrator interface on the CAS
         *
         */
        public com.cboe.idl.cmi.Administrator getAdministrator()
                      throws com.cboe.exceptions.SystemException,
                      com.cboe.exceptions.CommunicationException,
                      com.cboe.exceptions.AuthorizationException {
               if (administrator == null) {
                      administrator = getUserSessionManager().getAdministrator();
               }
               return administrator;
        }

        /**
         * Return the object reference for the MarketQuery interface on the CAS
         *
         */
        public com.cboe.idl.cmi.MarketQuery getMarketQuery()
                      throws com.cboe.exceptions.SystemException,
                      com.cboe.exceptions.CommunicationException,
                      com.cboe.exceptions.AuthorizationException {
               if (marketQuery == null) {
                      marketQuery = getUserSessionManager().getMarketQuery();
               }
               return marketQuery;
        }

        /**
         * Return the object reference for the OrderEntry interface on the CAS
         *
         */
        public com.cboe.idl.cmi.OrderEntry getOrderEntry()
                      throws com.cboe.exceptions.SystemException,
                      com.cboe.exceptions.CommunicationException,
                      com.cboe.exceptions.AuthorizationException {
               if (orderEntry == null) {
                orderEntry = getUserSessionManager().getOrderEntry();
            }
            return orderEntry;
        }

        public com.cboe.idl.cmiV3.OrderEntry getOrderEntryV3()
                throws com.cboe.exceptions.SystemException,
                com.cboe.exceptions.CommunicationException,
                com.cboe.exceptions.AuthorizationException {
            if (orderEntryV3 == null) {
                orderEntryV3 = getUserSessionManagerV3().getOrderEntryV3();
            }
            return orderEntryV3;
        }

        /**
         * Return the object reference for the OrderQuery interface on the CAS
         *
         */
        public com.cboe.idl.cmi.OrderQuery getOrderQuery()
                throws com.cboe.exceptions.SystemException,
                com.cboe.exceptions.CommunicationException,
                com.cboe.exceptions.AuthorizationException {
            if (orderQuery == null) {
                orderQuery = getUserSessionManager().getOrderQuery();
            }
            return orderQuery;
        }

        public com.cboe.idl.cmiV3.OrderQuery getOrderQueryV3()
                throws com.cboe.exceptions.SystemException,
                com.cboe.exceptions.CommunicationException,
                com.cboe.exceptions.AuthorizationException {
            if (orderQueryV3 == null) {
                orderQueryV3 = getUserSessionManagerV3().getOrderQueryV3();
            }
            return orderQueryV3;
        }

        /**
         * Return the object reference for the ProductDefinition interface on the
         * CAS
         *
         */
        public com.cboe.idl.cmi.ProductDefinition getProductDefinition()
                throws com.cboe.exceptions.SystemException,
                com.cboe.exceptions.CommunicationException,
                com.cboe.exceptions.AuthorizationException {
            if (productDefinition == null) {
                productDefinition = getUserSessionManager().getProductDefinition();
            }
            return productDefinition;
        }

        /**
         * Return the object reference for the ProductQuery interface on the CAS
         *
         */
        public com.cboe.idl.cmi.ProductQuery getProductQuery()
                throws com.cboe.exceptions.SystemException,
                com.cboe.exceptions.CommunicationException,
                com.cboe.exceptions.AuthorizationException {
            if (productQuery == null) {
                productQuery = getUserSessionManager().getProductQuery();
            }
            return productQuery;
        }

        /**
```

```
         * Return the object reference for the Quote interface on the CAS
         *
         */
        public com.cboe.idl.cmi.Quote getQuote()
                        throws com.cboe.exceptions.SystemException,
                        com.cboe.exceptions.CommunicationException,
                        com.cboe.exceptions.AuthorizationException {
                if (quote == null) {
                        quote = getUserSessionManager().getQuote();
                }
                return quote;
        }

        /**
         * Return the object reference for the TradingSession interface on the CAS
         *
         */
        public com.cboe.idl.cmi.TradingSession getTradingSession()
                        throws com.cboe.exceptions.SystemException,
                        com.cboe.exceptions.CommunicationException,
                        com.cboe.exceptions.AuthorizationException {
                if (tradingSession == null) {
                        tradingSession = getUserSessionManager().getTradingSession();
                }
                return tradingSession;
        }

        /**
         * Return the object reference for the UserPreferenceQuery interface on the
         * CAS
         *
         */
        public com.cboe.idl.cmi.UserPreferenceQuery getUserPreferenceQuery()
                        throws com.cboe.exceptions.SystemException,
                        com.cboe.exceptions.CommunicationException,
                        com.cboe.exceptions.AuthorizationException {
                if (userPreferenceQuery == null) {
                        userPreferenceQuery = getUserSessionManager()
                                        .getUserPreferenceQuery();
                }
                return userPreferenceQuery;
        }

        /**
         * Return object reference to UserTradingParameters service on the CAS.
         */
        public com.cboe.idl.cmi.UserTradingParameters getUserTradingParameters()
                        throws com.cboe.exceptions.SystemException,
                        com.cboe.exceptions.CommunicationException,
                        com.cboe.exceptions.AuthorizationException {
                if (userTradingParameters == null) {
                        userTradingParameters = getUserSessionManager()
                                        .getUserTradingParameters();
                }
                return userTradingParameters;
        }

        /**
         * Return object reference to UserHistory service on the CAS.
         */
        public com.cboe.idl.cmi.UserHistory getUserHistory()
                        throws com.cboe.exceptions.SystemException,
                        com.cboe.exceptions.CommunicationException,
                        com.cboe.exceptions.AuthorizationException {
                if (userHistory == null) {
                        userHistory = getUserSessionManager().getUserHistory();
                }
                return userHistory;
        }
        /**
         * Return the object reference for the Version interface on the CAS.
         *
         */
        public String getVersion() throws com.cboe.exceptions.SystemException,
                       com.cboe.exceptions.CommunicationException,
                       com.cboe.exceptions.AuthorizationException {
               if (casVersion == null) {
                       casVersion = getUserSessionManager().getVersion();
               }
               return casVersion;
        }

        /**
         * Logoff the CAS - reinitialize this instance of the CASAccessManager
         *
         */

        public void logoff() throws com.cboe.exceptions.SystemException,
                       com.cboe.exceptions.CommunicationException,
                       com.cboe.exceptions.AuthorizationException {

               getUserSessionManager().logout();
               clearReferences();

        }

        public void logoffV3() throws com.cboe.exceptions.SystemException,
                       com.cboe.exceptions.CommunicationException,
                       com.cboe.exceptions.AuthorizationException {

               getUserSessionManagerV3().logout();
               clearReferences();

        }

        public void logoffV4() throws com.cboe.exceptions.SystemException,
                       com.cboe.exceptions.CommunicationException,
                       com.cboe.exceptions.AuthorizationException {

               getUserSessionManagerV4().logout();
               clearReferences();

        }

        private void clearReferences() {
               //
               // Clear out the CAS interface references
               //

               administrator = null;
               userSessionManager = null;
               userAccess = null;
               orderQuery = null;
               marketQuery = null;
               orderEntry = null;
               tradingSession = null;
               productQuery = null;
               productDefinition = null;
               userPreferenceQuery = null;
               quote = null;
               casVersion = null;
               userAccessIOR = null;
               userTradingParameters = null;
               userHistory = null;
               userSessionAdminCallback = null;
               casIPAddress = null;
               casTCPPortNumber = 0;
```

```
                userAccessV2IOR = null;
                quoteV2 = null;
                marketQueryV2 = null;
                orderQueryV2 = null;
                userAccessV2 = null;
                sessionManagerStructV2 = null;
                userSessionManagerV2 = null;

                userAccessV3IOR = null;
                orderEntryV3 = null;
                orderQueryV3 = null;
                quoteV3 = null;
                marketQueryV3 = null;
                userAccessV3 = null;
                userSessionManagerV3 = null;

                userAccessV4 = null;
                userSessionManagerV4 = null;
        }

        /**
         * Provides the user with the Stringified IOR returned by the
         * UserAccessLocator object
         */
        public String getUserAccessIOR() {
                return userAccessIOR;
        }

        /**
         * Provides the user with the Stringified IOR returned by the
         * UserAccessLocator object
         */
        public String getUserAccessV2IOR() {
                return userAccessV2IOR;
        }

        /**
         * Provides the user with access to the CAS IP Address that was used to
         * access the CAS
         */
        public String getCASIPAddress() {
                return casIPAddress;
        }

        /**
         * Provides the user with access to the CAS TCP Port Number that was used to
         * access the CAS
         */
        public int getCASTCPPortNumber() {
                return casTCPPortNumber;
        }

        /**
         * Logon to the CAS
         *
         * @param userLogonStruct
         * @param userSessionAdminCallback
         *            Provide a constructed userSession admin callback object. The
         *            client is responsible
         * @param casIPAddress
         *            The IP Address or domain name of the CAS
         * @param casTCPPortNumber
         *            The TCP Port Number of the CAS HTTP Server
         */
        public void logonV2(
                        com.cboe.idl.cmiUser.UserLogonStruct userLogonStruct,
                        com.cboe.idl.cmiCallback.CMIUserSessionAdmin
userSessionAdminCallback,
                        String casIPAddress, int casTCPPortNumber)
                        throws com.cboe.exceptions.SystemException,
                        com.cboe.exceptions.CommunicationException,
                        com.cboe.exceptions.AuthorizationException,
                        com.cboe.exceptions.AuthenticationException,
                        com.cboe.exceptions.DataValidationException,
                        com.cboe.exceptions.NotFoundException {

                this.userSessionAdminCallback = userSessionAdminCallback;

                this.casIPAddress = casIPAddress;
                this.casTCPPortNumber = casTCPPortNumber;

                UserAccessLocator locator = new UserAccessLocator(casIPAddress,
                               casTCPPortNumber, USER_ACCESS_V2_REFERENCENAME);

                //
                // UserLocator.obtainIOR() will throw a
                // com.cboe.exceptions.CommunicationException
                // if it cannot talk to the CAS. This will be handled by the
                // CASAccessManager client.
                //
                this.userAccessV2IOR = locator.obtainIOR();

                org.omg.CORBA.Object objRef;

                objRef = orb.string_to_object(userAccessV2IOR);

                //
                // The UserAccessHelper Class is generated by the CORBA IDL
                // compiler for Java.
                //

                userAccessV2 = com.cboe.idl.cmiV2.UserAccessV2Helper.narrow(objRef);

                short sessionType = LoginSessionTypes.PRIMARY;
                //
                // Exceptions thrown by the UserAccess.logon() operation will be handled
                // by the CASAccessManager client.
                //
                sessionManagerStructV2 = userAccessV2.logon(userLogonStruct,
                               sessionType, userSessionAdminCallback, gmdTextMessaging);
                userSessionManagerV2 = sessionManagerStructV2.sessionManagerV2;
                userSessionManager = sessionManagerStructV2.sessionManager;
                return;
        }

        public void logonV2WhileAlreadyLoggedin(
                        com.cboe.idl.cmi.UserSessionManager userSessionManager)
                        throws com.cboe.exceptions.SystemException,
                        com.cboe.exceptions.CommunicationException,
                        com.cboe.exceptions.AuthorizationException,
                        com.cboe.exceptions.AuthenticationException,
                        com.cboe.exceptions.DataValidationException,
                        com.cboe.exceptions.NotFoundException {

                this.userSessionManager = userSessionManager;

                UserAccessLocator locator = new UserAccessLocator(casIPAddress,
                               casTCPPortNumber, USER_ACCESS_V2_REFERENCENAME);

                //
                // UserLocator.obtainIOR() will throw a
                // com.cboe.exceptions.CommunicationException
                // if it cannot talk to the CAS. This will be handled by the
                // CASAccessManager client.
                //
                this.userAccessV2IOR = locator.obtainIOR();
```

```
                org.omg.CORBA.Object objRef;

                objRef = orb.string_to_object(userAccessV2IOR);

                userAccessV2 = com.cboe.idl.cmiV2.UserAccessV2Helper.narrow(objRef);

                userSessionManagerV2 = userAccessV2
                               .getUserSessionManagerV2(this.userSessionManager);

                return;
        }

        /**
         * Return the object reference for the UserSessionManager interface on the
         * CAS
         *
         */
        public com.cboe.idl.cmiV2.UserSessionManagerV2 getUserSessionManagerV2()
                        throws com.cboe.exceptions.AuthorizationException {

                if (userSessionManagerV2 == null) {
                        com.cboe.exceptions.AuthorizationException authorizationException =
new com.cboe.exceptions.AuthorizationException();
                        short severity = 1;
                        authorizationException.details = new
com.cboe.exceptions.ExceptionDetails(
                                        "CASAccessManager Error: Trying to access
UserSessionManagerV2 before being logged on or after being logged off",
                                        new Date().toString(), severity, 9999);
                        throw authorizationException;
                }

                return userSessionManagerV2;
        }

        /**
         * Return the object reference for the MarketQuery interface on the CAS
         *
         */
        public com.cboe.idl.cmiV2.MarketQuery getMarketQueryV2()
                        throws com.cboe.exceptions.SystemException,
                        com.cboe.exceptions.CommunicationException,
                        com.cboe.exceptions.AuthorizationException {
                if (marketQueryV2 == null) {
                        marketQueryV2 = getUserSessionManagerV2().getMarketQueryV2();
                }
                return marketQueryV2;
        }

        /**
         * Return the object reference for the OrderQuery interface on the CAS
         *
         */
        public com.cboe.idl.cmiV2.OrderQuery getOrderQueryV2()
                        throws com.cboe.exceptions.SystemException,
                        com.cboe.exceptions.CommunicationException,
                        com.cboe.exceptions.AuthorizationException {
                if (orderQueryV2 == null) {
                        orderQueryV2 = getUserSessionManagerV2().getOrderQueryV2();
                }
                return orderQueryV2;
        }

        /**
         * Return the object reference for the Quote interface on the CAS
         *
         */
        public com.cboe.idl.cmiV2.Quote getQuoteV2()
                        throws com.cboe.exceptions.SystemException,
                        com.cboe.exceptions.CommunicationException,
                        com.cboe.exceptions.AuthorizationException {
                if (quoteV2 == null) {
                        quoteV2 = getUserSessionManagerV2().getQuoteV2();
                }
                return quoteV2;
        }

        public com.cboe.idl.cmiV3.Quote getQuoteV3()
                        throws com.cboe.exceptions.SystemException,
                        com.cboe.exceptions.CommunicationException,
                        com.cboe.exceptions.AuthorizationException {
                if (quoteV3 == null) {
                        quoteV3 = getUserSessionManagerV3().getQuoteV3();
                }
                return quoteV3;
        }

        public void logonV3(
                        com.cboe.idl.cmiUser.UserLogonStruct userLogonStruct,
                        com.cboe.idl.cmiCallback.CMIUserSessionAdmin
userSessionAdminCallback,
                        String casIPAddress, int casTCPPortNumber)
                        throws com.cboe.exceptions.SystemException,
                        com.cboe.exceptions.CommunicationException,
                        com.cboe.exceptions.AuthorizationException,
                        com.cboe.exceptions.AuthenticationException,
                        com.cboe.exceptions.DataValidationException,
                        com.cboe.exceptions.NotFoundException {

                this.userSessionAdminCallback = userSessionAdminCallback;

                this.casIPAddress = casIPAddress;
                this.casTCPPortNumber = casTCPPortNumber;

                UserAccessLocator locator = new UserAccessLocator(casIPAddress,
                                casTCPPortNumber, USER_ACCESS_V3_REFERENCENAME);

                //
                // UserLocator.obtainIOR() will throw a
                // com.cboe.exceptions.CommunicationException
                // if it cannot talk to the CAS. This will be handled by the
                // CASAccessManager client.
                //
                this.userAccessV3IOR = locator.obtainIOR();

                org.omg.CORBA.Object objRef;

                objRef = orb.string_to_object(userAccessV3IOR);

                //
                // The UserAccessHelper Class is generated by the CORBA IDL
                // compiler for Java.
                //

                userAccessV3 = com.cboe.idl.cmiV3.UserAccessV3Helper.narrow(objRef);

                short sessionType = LoginSessionTypes.PRIMARY;
                //
                // Exceptions thrown by the UserAccess.logon() operation will be handled
                // by the CASAccessManager client.
                //
                userSessionManagerV3 = userAccessV3.logon(userLogonStruct, sessionType,
                                userSessionAdminCallback, gmdTextMessaging);
                userSessionManagerV2 = userSessionManagerV3;
                userSessionManager = userSessionManagerV3;
                return;
        }
        public UserSessionManagerV3 getUserSessionManagerV3()
                        throws AuthorizationException {

                if (userSessionManagerV3 == null) {
                        System.out.println("You are not logon");
                        AuthorizationException authorizationException = new
AuthorizationException();
                        short severity = 1;
                        authorizationException.details = new ExceptionDetails(
                                        "CASAccessManager Error: Trying to access
UserSessionManagerV3 before being logged on or after being logged off",
                                        new Date().toString(), severity, 9999);
                        throw authorizationException;
                }

                return userSessionManagerV3;
        }

        public com.cboe.idl.cmiV3.MarketQuery getMarketQueryV3()
                        throws SystemException, CommunicationException,
                        AuthorizationException {
                if (marketQueryV3 == null) {
                        marketQueryV3 = getUserSessionManagerV3().getMarketQueryV3();
                }
                return marketQueryV3;
        }

        public com.cboe.idl.cmiV4.MarketQuery getMarketQueryV4()
                        throws SystemException, CommunicationException,
                        AuthorizationException {
                if (marketQueryV4 == null) {
                        marketQueryV4 = getUserSessionManagerV4().getMarketQueryV4();
                }
                return marketQueryV4;
        }

        public void logonV4(
                        com.cboe.idl.cmiUser.UserLogonStruct userLogonStruct,
                        com.cboe.idl.cmiCallback.CMIUserSessionAdmin
userSessionAdminCallback,
                        String casIPAddress, int casTCPPortNumber)
                        throws com.cboe.exceptions.SystemException,
                        com.cboe.exceptions.CommunicationException,
                        com.cboe.exceptions.AuthorizationException,
                        com.cboe.exceptions.AuthenticationException,
                        com.cboe.exceptions.DataValidationException,
                        com.cboe.exceptions.NotFoundException {

                this.userSessionAdminCallback = userSessionAdminCallback;

                this.casIPAddress = casIPAddress;
                this.casTCPPortNumber = casTCPPortNumber;

                UserAccessLocator locator = new UserAccessLocator(casIPAddress,
                                casTCPPortNumber, USER_ACCESS_V4_REFERENCENAME);

                //
                // UserLocator.obtainIOR() will throw a
                // com.cboe.exceptions.CommunicationException
                // if it cannot talk to the CAS. This will be handled by the
                // CASAccessManager client.
                //
                this.userAccessV4IOR = locator.obtainIOR();

                org.omg.CORBA.Object objRef;

                objRef = orb.string_to_object(userAccessV4IOR);

                //
                // The UserAccessHelper Class is generated by the CORBA IDL
                // compiler for Java.
                //

                userAccessV4 = com.cboe.idl.cmiV4.UserAccessV4Helper.narrow(objRef);

                short sessionType = LoginSessionTypes.PRIMARY;
                //
                // Exceptions thrown by the UserAccess.logon() operation will be handled
                // by the CASAccessManager client.
                //
                userSessionManagerV4 = userAccessV4.logon(userLogonStruct, sessionType,
                               userSessionAdminCallback, gmdTextMessaging);
                userSessionManagerV3 = userSessionManagerV4;
                userSessionManagerV2 = userSessionManagerV4;
                userSessionManager = userSessionManagerV4;
                return;
        }

        public UserSessionManagerV4 getUserSessionManagerV4()
                        throws AuthorizationException {

                if (userSessionManagerV4 == null) {
                        AuthorizationException authorizationException = new
AuthorizationException();
                        short severity = 1;
                        authorizationException.details = new ExceptionDetails(
                                        "CASAccessManager Error: Trying to access
UserSessionManagerV4 before being logged on or after being logged off",
                                        new Date().toString(), severity, 9999);
                        throw authorizationException;
                }

                return userSessionManagerV4;
        }

}
```

Downloading Class Information

After logging onto the CAS the next step is to download and cache class information for use in subscribing to the different data feeds. This is a query operation that returns a snapshot of class information. For instance, if you wanted a list of all classes of a certain type you would invoke the following operation on the ProductQuery interface:

ProductQuery::getProductClasses(productType) which returns a sequence of cmiProduct::ClassStructs

Where:

productType is a cmiProduct::ProductType that indicates what type of classes are to be returned (ProductTypes.OPTIONS, ProductTypes.EQUITY, ProductTypes.FUTURE, etc).

See the function getProductClassesByType in com.cboe.examples.example15.example15 or example15.cpp to see how this is called.

Subscribing to Class Data

First, create callback objects that implement the following interfaces: cmiCallbackV4::CMICurrentMarketConsumer for current market data, cmiCallbackV4::CMITickerConsumer for ticker data, and cmiCallbackV4::CMIRecapConsumer for recap and last sale data.

See com.cboe.examples.example15.CurrentMarketV4ConsumerCallback or CurrentMarketV4Callback.cpp, com.cboe.examples.example15.RecapV4ConsumerCallback or RecapV4ConsumerCallback.cpp, com.cboe.examples.example15.TickerV4ConsumerCallback or RecapV4ConsumerCallback.cpp as examples.

> **Note**: CBOE strongly recommends implementing the callbacks in separate objects.

Next, register these callbacks with your orb so that the CAS can call them as needed. See the Example 15 program for a specific example.

Finally, subscribe to a class as needed using the MarketQueryV4 interface. For instance, if you want to subscribe for ticker data for a particular class, invoke the operation:

MarketQueryV4::subscribeTicker(classKey,clientListener,actionOnQueue);

Where:

- classKey is a cmiProduct::ClassKey that specifies the class whose ticker data you want to receive.
- clientListener is the cmiCallbackV4::CMITickerConsumer object you have implemented within your program to process ticker information.
- actionOnQueue is the cmiUtil::QueueAction value that indicates what the desired queue behavior you would like on the CAS.

The operation that the CAS will invoke on your CMITickerConsumer is:

acceptTicker(tickerStructs,messageSequence,queueDepth,queueAction)

Where:

- tickerStructs is a cmiMarketData::TickerStructV4Sequence which is a sequence of cmiMarketData::TickerStructs
- messageSequence is a CORBA long that indicates the order in which the message was sent
- queueDepth is a CORA long that indicated the depth of the queue on the CAS
- queueAction is a cmiUtil::QueueAction that describes the action taken on the queue, if any

To unsubscribe a given class key from the CAS call the corresponding unsubscribe function in the cmiV4::MarketQueryV4 interface. For example, to unsubscribe from getting ticker data for a specific class key call:

MarketQueryV4::unsubscribeTicker(classKey,clientListener);

Where:

- classKey is a cmiProduct::ClassKey that specifies the class whose ticker data you no longer want to receive.
- clientListener is the cmiCallbackV4::CMITickerConsumer in your application that processes tickers for the avove class key.

Below is a list of the MDX market data callbacks:

CMi MDX Callbacks
cmiCallbackV4::CMICurrentMarketConsumer oneway CORBA callback interface to receive current market data.
cmiCallbackV4::CMIRecapConsumer oneway CORBA callback interface to receive recap and last sale information.
cmiCallbackV4::CMITickerConsumer oneway CORBA callback interface to receive ticker information.

Callback Performance Design Issues

Threading is an extremely important aspect of callbacks both on the CAS and client application. Instances of callback objects are registered with the CMi to receive current market data, e.g. cmiCallbackV4::CMICurrentMarketConsumer is registered as a call back by calling:

> cmiV4::MarketQuery.subscribeCurrentMarket(in cmiProduct::ClassKey classKey, in cmiCallbackV4::CMICurrentMarketConsumer clientListener, in cmiUtil::QueueAction actionOnQueue)

The CAS associates a thread and a queue with each unique callback object. Each queue is processed independently. The CAS delivers the data to each callback in the order that it’s received. With an ordinary CORBA callback this would guarantee message delivery to a callback object in the order that they were received (by the CAS). However, the MDX data is delivered by a significantly different callback mechanism.

Oneway CORBA Calls

The cmiCallbackV4 callbacks are implemented as oneway CORBA calls. These are high-performance calls that function differently than a standard CORBA call. Normally a CORBA function call will wait for the call to complete executing and return a return value (if any), i.e. it waits for the full round trip to complete before continuing. A oneway CORBA call will not wait for any return from the function call before continuing. In other words, the CAS calls your oneway callback object’s acceptXXX() method and won’t wait for it to return before continuing its execution.

Regardless of client implementation, the CAS will still process data sequentially for a given callback object’s queue. However, the client application’s POA theading policy will dictate how the client receives the data. If the client application’s POA is single-threaded then things will work as they were before with data arriving in the expected sequential order. If the POA policy allows more than one thread per POA then there is no guarantee what order the data will be received. Due to the CORBA oneway callback it’s extremely likely that two or more POA threads may concurrently call the same callback object’s acceptXXX() function.

To help with this problem a sequence number is provided in the callback method signatures as an aid to determine whether a message was received out of order. It is essential to keep track of the current sequence number and compare incoming messages to it regardless of POA threading policy. If you are using the OVERLAY mode, you may receive data structures for more than one derivative product per message. The sequence number should be associated with all the contained data structures in the message. You would need to disregard only the individual product data structures that are out of order and not the entire contents of the message. More detail is in Example 15 but a simplified example would be:

Give we have options product keys 100, 101, 102, 103, 104, 105.

Message #7 has data for 101, 102, 105

- Firm application stores 101/#7, 102/#7, 105/#7

Message #10 has data for 104, 105, 106, 107

- Firm application stores 104/#10, 105/#10, 106/#10, 107/#10

Message #8 has data for 100, 101, 102, 103

- Firm application stores 100/#8, 101/#8, 102/#8, 103/#8

Message #9 has data for 102, 103, 104, 105, 106

- Firm application stores 102/#9, 103/#9

- Firm application sees it already newer entries (#10) for 104, 105, 106

CBOE recommends that each class has its own callback object, especially for high volume classes like GOOG, IBM, MSFT, etc. In theory, the client could use the same callback object for several classes. For example, the client application could instantiate a callback object, and subscribe for products for the classes IBM, GE, and ORCL using just that callback. If the callback in the client application is activated with a multithreaded POA, it has the ability to handle concurrent calls for all three classes. However, since the CAS sees this as a single callback, and associates the callback with a single queue, the calls would actually be serialized in the CAS and the potential for queueing much greater. This is a particularly bad choice for high volumes of messages, especially as will be found in options trading.

> **Note**: if one callback object is registered for multiple classes then the sequence numbers it receives are specific to that particular class. The callback object then must keep track of the current sequence number by class as well.

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

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

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

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

- The CAS associates a thread and a queue with each unique callback.
- An excessive number of threads can affect performance due to the overhead of managing the threads
- Too few callbacks can affect performance due to the serialization of messages to a single callback.

If your implementation associates multiple callback objects to a single servant instance, it is important that the servant be thread safe and not contain any code that could inadvertently synchronize the threads against each other.

When you register a callback object you become a CORBA server. It is important to make sure that your Orb has been configured with enough threads in its POA threadpool to service all the incoming messages. One performance aspect often overlooked is the available CORBA ORB resources. Depending on your ORB the default setting can be limiting. Review your ORB documentation for limitations with regard to thread usage and adjust the settings to your use. While we do not provide a performance lab it is simple enough to create a simulator to drive your ORB (and your own application) to assist in the tuning.

Another consideration is to limit the processing on each callback thread to the absolute minimum. The faster a callback returns the faster the CAS can send the next message for that subscription. The CAS message performance is directly related to how efficient the client listeners are.

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

The CAS provides a Queue Depth parameter on the callbacks. This value will inform you of the depth of any queue your CAS server currently has for your process. You will be able to set a threshold and an action to take to eliminate the queue.

Market Data Publishing

The MDX interfaces return a sequence of updates for a given class in one message. All methods take advantage of blocking where appropriate. For instance, a single block of quotes from any user can result in multiple updates to the current market quote for several products within a class. CBOE will deliver the market updates in a single message where possible. Below is a summary of MDX market data types and when they are published.

Market Data Type	**Description**	**Trading Sessions**	**Request Type**	**Subscription Type**	**When Published**
Current Market	Bid Price Ask Price Bid Size Ask Size Market Indicator Product State	N/A	Subscription plus initial snapshot at time of subscription	Class Only	Anytime there is a change in the fields referenced to the left
Recap	High Price Low Price Open Price Close Price	N/A	Subscription plus initial snapshot at time of subscription	Class Only	Anytime there is a change in the high, low, open, close price
Last Sale	Last sale for underlying and derivative products Last Sale Price Last Sale Volume Total Volume Direction Net Price Change	N/A	Subscription plus initial snapshot at time of subscription	Class Only	On every trade
Ticker	Record sent for latest trade Trade Price Trade Volume Trade Time Sale Prefix/Postfix	N/A	At subscription	Class Only	On every trade

Market Data Struct Definitions

The following tables define the MDX structure.

CurrentMarketStructV4

Attribute	**Type**	**Documentation**
askPrice	IntegerPrice	The ask price
askSizeSequence	MarketVolumeStructSequence	Sequence of MarketVolumeStructs that contain the order volume broken down by order contingency type at the current market.
bidPrice	IntegerPrice	The bid price
bidTickDirection	TickDirectionType	The direction of change for the bid PLUS_TICK '+' Means the current last sale price is higher than the previous last sale price. Also known as an "uptick". MINUS_TICK '-' Means the current last sale price is lower than the previous last sale price. Also known as a "downtick". ZERO_MINUS_TICK '_' Means the current last sale price is the same as the previous last sale price, but is lower than the most recent different last sale price. ZERO_PLUS_TICK '*' Means the current last sale price is the same as the previous last sale price, but is higher than the most recent different last sale price. UNKNOWN_TICK ' ' Means the tick direction field is not used or not known.
bidSizeSequence	MarketVolumeStructSequence	Sequence of MarketVolumeStructs that contain the order volume broken down by order contingency type at the current market.
classKey	ClassKey	Class key for the product whose current market is

Attribute	Type	Documentation
		being reported
currentMarketType	CurrentMarketType	The market type, either best market or best public market
exchange	string	Name of the exchange. Your account has to be enabled by CBOE help desk for getting away market data. If you are not enabled, you will still only get CBOE data.
marketIndicator	MarketIndicator	The market indicator for this product. See file cmiConstants.idl
priceScale	octet	The number of decimal places to use in conjunction with the integer prices
productKey	ProductKey	Product key for the product whose current market is being reported
productState	ProductState	The product state for this product
productType	ProductType	The product type (equity, option, etc.)
sentTime	IntegerTime	Time that the information was sent

LastSaleStructV4

Attribute	Type	Documentation
classKey	ClassKey	Class key for the product whose current market is being reported
tickDirection	char	The direction of change in current last sale price relative to the previous last sale. PLUS_TICK '+' Means the current last sale price is higher than the previous last sale price. Also known as an "uptick". MINUS_TICK '-' Means the current last sale price is lower than the previous last sale price. Also known as a "downtick". ZERO_MINUS_TICK '_' Means the current last sale price is the same as the previous last sale price, but is lower than the most recent different last sale price.

Attribute	Type	Documentation
		ZERO_PLUS_TICK '*' Means the current last sale price is the same as the previous last sale price, but is higher than the most recent different last sale price. UNKNOWN_TICK ' ' Means the tick direction field is not used or not known.
exchange	string	Name of the exchange. Your account has to be enabled by CBOE help desk for getting away market data. If you are not enabled, you will still only get CBOE data.
lastSalePrice	IntegerPrice	The last sale price Integer form
lastSaleTime	IntegerTime	The time of the last sale (milliseconds since midnight)
lastSaleVolume	long	The volume of the last sale
netPriceChange	IntegerPrice	Changes from previous close price. For CBOE option, it is the change from first last sale of the day. This is a **signed** value as well and can therefore be a positive, zero, or negative IntegerPrice value. Example: CurrentLastSale: 10.00 YesterdayClose: 10.05 ==> NetChange == - 0.05 (represented as integer -5 with a scale of 2).
priceScale	octet	The number of decimal places to use in conjunction with the integer prices
productKey	ProductKey	Product key for the product whose current market is being reported
productType	ProductType	The product type (equity, option, etc.)
totalVolume	long	The total volume for this product

RecapStructV4

Attribute	Type	Documentation
classKey	ClassKey	Class key for the product

Attribute	Type	Documentation
		whose current market is being reported
exchange	string	Name of the exchange. Your account has to be enabled by CBOE help desk for getting away market data. If you are not enabled, you will still only get CBOE data.
highPrice	IntegerPrice	Today's high price
lowPrice	IntegerPrice	Today's low price
openPrice	IntegerPrice	Today's opening price
previousClosePrice	IntegerPrice	Previous closing price
priceScale	octet	The number of decimal places to use in conjunction with the integer prices
productKey	ProductKey	Product key for the product whose current market is being reported
productType	ProductType	The product type (equity, option, etc.)
statusCodes	string	Currently only used for equities, not options. Indicates whether there is Dow Jones News, Reuters News, whether the stock is Ex-Dividend, and whether it's opened or closed. "RN": indicates there is Reuters News for the product. "DJ": indicates there is Dow Jones News for the product. "XD": indicates it is in ex-dividend (the day dividends are paid out). For Options, this field is set to "" (empty String).

TickerStructV4

Attribute	Type	Documentation
classKey	ClassKey	Class key for the product whose current market is being reported
exchange	string	Name of the exchange. Your account has to be enabled by CBOE help desk for getting away market data.

Attribute	Type	Documentation
		If you are not enabled, you will still only get CBOE data.
lastSaleVolume	long	The volume of the last sale
priceScale	octet	The number of decimal places to use in conjunction with the integer prices
productKey	ProductKey	Product key for the product whose current market is being reported
productType	ProductType	The product type (equity, option, etc.)
salePostfix	string	Suffix information that is appended to the end of the ticker information by the reporting market – (*see the Sale Conditions section below for definitions)*
salePrefix	string	Prefix information provided by the market reporting the sale. These will be prefixed with '.' (dot). .OPNT (Opening Trade) is used if the first trade is part of the opening rotation. .FTAO (First Trade After Opening) is used if the first trade occurs after the opening. .CNCO (Cancel Opening) is a legitimate code for OPRA and is issued for a bust that affects the opening trade.
tradePrice	IntegerPrice	The trade price
tradeTime	IntegerTime	The time of the trade (milliseconds since midnight)
tradeVolume	Long	The volume for this trade

Sale Conditions

Option Sale Conditions

Suffix	**Definition**
(none)	Regular sale. Indicates that the transaction was a regular sale and was made without stated conditions.
.CANC	Cancel prior trade, not last sale, not opening trade. Transaction previously

Suffix	Definition
	reported (other than as the last or opening report for the particular option contract) is now to be cancelled.
.OSEQ	Late and out of sequence. Transaction is being reported late and is out of sequence; i.e., later transactions have been reported for the particular option contract.
.CNCL	Cancel last sale (not the opening trade). Transaction is the last reported for the particular option contract and is now cancelled.
.LATE	Late, but in sequence. Transaction is being reported late, but is in the correct sequence; i.e., no later transactions have been reported for the particular option contract.
.CNCO	Cancel opening trade, but other trades have occurred. Transaction was the first one (opening) reported this day for the particular option contract. Although later transactions have been reported, this transaction is now to be cancelled.
.OPEN	Late report of opening trade, out of sequence. Transaction is a late report of the opening trade and is out of sequence; i.e., other transactions have been reported for the particular option contract.
.CNOL	Cancel Only / Open. Product should go closed. Transaction was the only one reported this day for the particular option contract and is now to be cancelled.
.OPNL	Late report of opening trade, in sequence. Transaction is a late report of the opening trade, but is in the correct sequence; i.e., no other transactions have been reported for the particular option contract.
.AUTO	Trade was executed automatically. Transaction was executed electronically. This appears solely for information; process as a regular transaction.
.REOP	Reopen halted product. Transaction is a reopening of an option contract in which trading has been previously halted. This appears solely for information; process as a regular transaction.
.AJST	Contract terms non-standard or adjusted. Transaction is an option contract for which the terms have been adjusted to reflect a stock dividend, stock split, or similar event. This appears solely for information; process as a regular transaction.
.SPRD	Spread trade. Transaction represents a trade in two options in the same class (a buy and a sell in the same class). This appears solely for information; process as a regular transaction.
.STDL	Straddle. Transaction represents a trade in two options in the same class (a buy and a sell in a put and a call). This appears solely for information; process as a regular transaction.
.STPD	Stopped trade. Transaction is the execution of a sale at a price agreed upon by the floor personnel involved, where a condition of the trade is that it reported following a non-stopped trade of the same series at the same price.
.CSTP	Cancel stopped trade. Cancel stopped transaction.
.BWRT	Buy Write. Transaction represents the option portion of an order involving a single option leg (buy or sell of a call or put) and stock. This appears solely for information: process as a regular transaction.
.CMBO	Combination Spread. Transaction represents the buying of a call and the selling of

Suffix	Definition
	a put for the same underlying stock or index. This appears solely for information; process as a regular transaction.
.BLCK	Blocked Trade. Not documented in the OPRA specifications.
.EFPT	Exchange for physical. Not documented in the OPRA specifications.

Equity Sale Conditions

Suffix	Definition
(none)	Regular sale. Indicates that the transaction was a regular sale and was made without stated conditions
.CSHO	Cash only market. A security settling in cash all day on a participant or consolidated basis, such as a Common, Preferred or Right that is nearing expiration. Settlement is similar to a Cash Trade, except that Cash (only) Market trades qualify to update a security’s trading range (high, low, last calculations) during the day. Participants can elect to report different settlements in the same security during the day based on their own settlement requirements. For example, one participant can report trades as cash (only) market while another participant can report trades as regular or next day settlement. For Network B bonds, cash (only) market can be used to report transactions in a regular way market.
.AVGP	SIAC average price trade. A trade where the price reported is based upon an average of the prices for transactions in a security during all or any portion of the trading day.
.CSH	Cash sale. A transaction that calls for the delivery of securities and payment on the same day the trade took place.
.NDO	Next day only market. Same definitions as cash market except settlement is next day.
.IMSWP	Intermarket sweep order. A sale condition that indicates to CTS Data Recipients that the execution price reflects the order instruction not to send the order to another market that may have had a superior price.
.BKCLS	Basket index on close. A basket index on close transaction signifies a trade involving paired basket orders, the execution of which is based on the closing value of the index. These trades are reported after the close when the index closing value is determined.
.R127	Rule 127 trade (NYSE only). To qualify as a 127 print the trade is executed outside the present quote and meets one or both of the following conditions: (1) has a volume of 10,000 shares or more and/or (2) has a dollar value of $200,000 or more.
.R155	Rule 155 trade (AMEX only). To qualify as a 155 print, a specialist arranges for the sale of the block at one "clean-up" price or at the different price limits on his book. If the block is sold at a "clean-up" price, the specialist should execute at the same price all the executable buy orders on his book. This Sale Condition is only applicable for AMEX trades.
.SLST	Sold last. Sold Last is used when a trade prints in sequence but is reported late or printed in conformance to the One or Two Point Rule.
.ND	Next day. A transaction that calls for the delivery of securities between one and four days (to be agreed by both

Suffix	Definition
	parties to the trade - the number of days are not noted with the transaction) after the trade date.
.OPN	Opened. Indicates an opening transaction that is printed out of sequence and at the wrong time.
.SEL	Seller. A Seller's option transaction is a special transaction that gives the seller the right to deliver the stock at any time within a specific period, ranging from not less than four calendar days to not more than sixty calendar days.
.AUTO	Automatic execution. A sale condition code that identifies a NYSE trade that has been automatically executed without the potential benefit of price improvement.
.SLD	Sold out of sequence. Sold Out of Sequence is used when a trade is printed (reported) out of sequence and at a time different from the actual transaction time.
.AQ	Acquisition. A transaction made on the Exchange as a result of an Exchange acquisition.
.BCH	Bunched trade. A trade representing an aggregate of two or more regular trades in a security occurring at the same price either simultaneously or within the same 60-second period, with no individual trade exceeding 10,000 shares.
.BCHSLD	Bunched sold trade. A bunched trade that is reported late.
.DIST	Distribution. Sale of a large block of stock in such a manner that the price is not adversely affected.
.SPL	Split trade. An execution in two markets when the specialist or Market Maker in the market first receiving the order agrees to execute a portion of it at whatever price is realized in another market to which the balance of the order is forwarded for execution.
.PMKTCL	Form T post-market trade. A trade executed before or after the regular US market hours. Please note that the Dot-T modifier will be appended to all transactions that occur during the pre- and post-market sessions. The volume of Form-T trades will be included in the calculation of consolidated and market center volume. The price information in Dot-T trades will not be used to update high, low and last sale data for individual securities or indices since they occur outside of normal trade reporting hours.
.AVGP	NASDAQ average price trade. A trade where the price reported is based upon an average of the prices for transactions in a security during all or any portion of the trading day. Please note that the NASDAQ market center also uses this value to report stopped stock situations.
.PRIOR	Prior reference price trade. An executed trade that relates to an obligation to trade at an earlier point in the trading day or that refers to a prior referenced price. This may be the result of an order that was lost or misplaced or was not executed on a timely basis.
.CPRV	Cancel prior trade, not last. Transaction previously reported (other than as the last or opening report for the particular contract) is now to be cancelled.
.LOSQ	Late trade, out of sequence. Transaction is being reported late and is out of sequence; i.e., later transactions have been reported for the

Suffix	Definition
	particular contract.
.CLST	Cancel last trade. Transaction is the last reported for the particular contract and is now cancelled.
.LATE	Late trade, in sequence. Transaction is being reported late, but is in the correct sequence; i.e., no later transactions have been reported for the particular contract.
.CNOP	Cancel only trade, not last. Transaction was the first one (opening) reported this day for the particular contract. Although later transactions have been reported, this transaction is now to be cancelled.
.CNOL	Cancel only trade. Transaction was the only one reported this day for the particular contract and is now to be cancelled.
.LOPN	Late open trade, in sequence. Transaction is a late report of the opening trade, but is in the correct sequence; i.e., no other transactions have been reported for the particular contract.
.REOP	Reopen halted. Transaction is a reopening of a contract in which trading has been previously halted. Prefix appears solely for information; process as a regular transaction.
.SPRD	Spread trade. Transaction represents a trade in two contract months in the same class (a buy and a sell in the same class). Prefix appears solely for information; process as a regular transaction.
.STDL	Straddle trade. Transaction represents a trade in two options in the same class (a buy and a sell in a put and a call). Prefix appears solely for information; process as a regular transaction.
.STPD	Stopped trade. Transaction is the execution of a sale at a price agreed upon by the floor personnel involved, where a condition of the trade is that it reported following a non-stopped trade of the same series at the same price.
.CSTP	Cancel stopped trade. Cancel stopped transaction.
.BRWT	Buy Write trade. Transaction represents the option portion of a buy/write (buy stock, sell call options). Prefix appears solely for information; process as a regular transaction.
.CMBO	Combination spread trade. Transaction represents the buying of a call and the selling of a put for the same underlying stock or index. Prefix appears solely for information; process as a regular transaction.
.BLKT	Block trade. A large amount of securities being traded, typically at least 10,000 shares of stock or $200,000 in bonds.
.EXPH	Exchange future for physical trade. Not documented in the CFN (ONE Vendor) specifications.
.CLSPRC	Market center close price. Indicates the "Official" closing value as determined by a Market Center. This transaction report will contain the market center generated closing price. This sale condition modifier shall only affect the Market Center Closing/Last Sale value and will not affect the

Suffix	Definition
	consolidated market value.
.OPRC	Market center open price. Indicates the "Official" opening value as determined by a Market Center. This transaction report will contain the market center generated opening price. This sale condition modifier shall only affect the Market Center Opening value and will not affect the consolidated market value. Direct data recipients that maintain individual market center open values should use this value as the official market center opening value and populate data displays accordingly
.STPREG	Stopped stock, regular trade.
.STPSLST	Stopped stock, sold last.
.STPSLD	Stopped stock, sold out of sequence. In accordance with AMEX Rule 109, a "stopped stock" transaction may occur under several circumstances, including when an AMEX Specialist executes market-at-the-close orders in a stock, where the Specialist is holding simultaneously both buy and sell market-at-the-close orders. The Specialist is required, under section (d) of the rule, to report the "pair off" transaction as "stopped stock". In addition a "stopped stock" transaction may occur when a Broker, trying to get a better price for the customer's market order than the currently available price, asks the Specialist to "stop the stock". The Specialist guarantees the Broker the current "stopped" price but does not immediately execute the order. The order is used by the Specialist to improve the quote in order to obtain a better price. If the next trade is at the "stopped" price, the order is "elected" and executed by the Specialist at the stopped price rather than at an improved price. The execution at the stopped price is designated as "Stopped Stock". Depending on the timing of the trade report message, one of three sale condition modifiers may be used to identify a stopped stock transaction: 1 = Stopped Stock - Regular Trade 2 = Stopped Stock - Sold Last 3 = Stopped Stock - Sold Out of Sequence
.SLST	NASDAQ sold last. Sold Last sale condition modifier is used when a trade prints in sequence but is reported late OR the trade is printed by AMEX in conformance to the One or Two Point Rule. A Sold Last transaction should only impact the consolidated last sale price for an issue if the market center reporting the sold last transaction also reported the transaction setting the current last sale price.
.EOSQ	NASDAQ extended trading hours, sold out of sequence. Trade reports used to identify extended trading hours trades that are reported more than 90 seconds after execution. Currently, the extended trading hours are comprised of pre-market trading from 8 a.m. to 9:30 a.m., Eastern Time (ET), and post-market trading from 4 p.m. to 6:30 p.m., ET. This sale condition would be similar to the existing "PMKTCL" sale condition in that trades executed outside of market hours will not impact market center or consolidated high, low, or last sale prices for an issue. The transactions would, however, count toward issue and market volume.
.YLFL	Yellow flag regular trade. Market Centers will have the ability to identify regular trades being reported during specific events as out of the ordinary by appending a new sale condition code Yellow Flag ("Y") on each transaction reported to the NASDAQ SIP. The new sale condition ".YLFL" will be eligible

Suffix	Definition
	to update all market center and consolidated statistics. In certain instances, the SIP will be required to append the ".YLFL" for the market center for trades reported as regular -way (no sale condition modifier suffix).

Queuing on the CAS

During heavy market activity there may be queuing on the CAS if the client application can’t keep up with the data flow. The CAS supports several methods of dealing with queueing:

(1) NO_ACTION - CBOE currently does not support this designation. It will return the same resulting actions as DISCONNECT_CONSUMER.

(2) FLUSH_QUEUE - Allows the end user to continue building large queues on the CAS. Once the queue reaches a certain size, it will automatically be flushed then allowed to rebuild. The queue size is currently configured to 5,000.

(3) DISCONNECT_CONSUMER - Automatically unregisters the callback object when the queue size is exceeded. Currently, the queue size value for the CAS is configured to 5,000.

(4) OVERLAY_LAST - The callback object receives the most recent market data information for a product, overlaying updates that are in progress. Below is an example of overlay mode.

CBOE recommends either the FLUSH_QUEUE or OVERLAY_LAST modes.

Overlay Mode Example

Let's say there are 10 products for a given class.

1) The firm gets an update from CBOE for one of those products.

2) Before the firm finishes processing that one update, the other nine tick.

3) The next call the firm gets from us will be the remaining nine ticks on the remaining nine products all at once.

So, one of the features of overlay is to "group" up Current Market into portions. The portions will be small if the firm processes them fast and potentially larger as the firm's processing becomes slower.

The second feature of overlay is to only give the very latest Current Market.

1) The firm gets an update from CBOE for product X

2) While the firm is processing that update, Product X ticks eight more times.

3) CBOE would then send only the eighth of those updates for Product X to the firm.

CBOE would not send the first seven messages in step #2 above. The only messages that will ever be dropped in overlay mode are "old" current market messages that are no longer "current" for one particular product. Those old current markets will be overlaid on a product by product basis. Also, there will never be queuing in overlay because of this, thus it is impossible to disconnect because of large queues if the firm is using overlay mode.