FeedOS C# API UserGuide

Quant House

July 29, 2011

Contents

1 About this document

This document aims to be a guide for first time users of FeedOS C# API. It provides step by step instructions on how to get started with C# API to connect, issue simple requests and subscribe to different level of market data.

Intended audience

This document is intended for Software Engineers and Quantitative Traders to be used.

Prerequisite

Microsoft .NET: .NET must be installed on the users machine. Please refer to release notes for .NET version compatibility.

Financial Knowledge/Jargon: Knowledge of financial instruments and their basic characteristics are assumed. Common financial jargon is used throughout this text.

Non Disclosure

2 Overview

This document gives an overview of FeedOS C# client API.

This document can be downloaded from http://www.quanthouse.com/download/#documentation.

The current FeedOS C# client API is built on top of the FeedOS C++ client API, and many common concepts are explained in the FeedOS C++ API User Guide available at the same place.

3 Initializing and completion of the API

How to initialize and complete the FeedOS C# API ?

3.1 Initializing the API

Prior using the API, you must call:

```
FeedOSManaged.API.Init(string applicationName);
```

Internal API logs settings have to be initialized.

```
FeedOSManaged.API.InitTraceAllAreas(
string logFileName,
bool appendMode,
bool flushEveryLine,
bool enableWarningLevel,
bool enableInfoLevel,
bool enableDebugLevel,
bool enableScopeLevel);
```

3.2 Completion of the API

When you are done using the API, you should call:

```
FeedOSManaged.API.Shutdown();
```

4 Connecting to a server

How to connect to a FeedOS server

4.1 Connection object

In order to communicate with a FeedOS server, you have to instantiate an object from the FeedOSManaged.Connection class. You may instantiate as many objects as needed, if reaching several servers at the same time makes sense.

4.2 Initiating the connection

You have to call method Connect on the target Connection object :

```
uint Connect (
string hostname,
uint port,
string username,
string password);
```

Expected parameters :

hostname : address (name or IP) of the server

port : TCP port of the FeedOS server

username : your username, as provided by FeedOS support personnel

password : your password

This call is synchronous. The return code tells if the connection was successful.

4.3 Closing the connection

You have to call method Disconnect on the target Connection object :

```
uint Disconnect();
```

4.4 Listening the FeedStatus

In order to subscribe to the FeedStatus events, you have to call method SubscribeToFeedStatus on the target Connection object :

```
void SubscribeToFeedStatus (unsigned int requestId );
```

see FeedStatusSnapshotEventHandler, FeedStatusUpdateEventHandler, FeedStatusModifiedEventHandler

4.5 Connection event handlers

Several event handlers may be registered to get information about connection

- public delegate void DisconnectedEventHandler(uint returnCode);
  public event DisconnectedEventHandler Disconnected;

Raised when disconnecting, voluntarily or not

- public delegate void AdminMessageEventHandler(bool isUrgent, string origin, string headline, string content);
  public event AdminMessageEventHandler AdminMessage;

Raised when a message is sent by the administrator of the FeedOS server

- public delegate void MsgDispatchingHookBeginEventHandler(uint nbPendingMsg);
  public event MsgDispatchingHookBeginEventHandler MsgDispatchingHookBegin;

Raised to give the number of messages which are going to be dispatched

- public delegate void MsgDispatchingHookEndEventHandler();
  public event MsgDispatchingHookEndEventHandler MsgDispatchingHookEnd;

Raised when all the messages cited above have been dispatched

- public delegate void FeedStatusSnapshotEventHandler(unsigned int requestId, List<FeedStatus> snapshotStatus); event FeedStatusSnapshotEventHandler FeedStatusSnapshot;

Raised at connection time to the FeedOS server, describing the FeedStatus initial snapshot

- public delegate void FeedStatusUpdateEventHandler(unsigned int requestId, string sender, DateTime serverUTCDateTime, FeedStatus status);
  event FeedStatusUpdateEventHandler FeedStatusUpdate;

Raised when the FeedStatus of the FeedOS server change, information is sent in detailed format

- public delegate void FeedStatusModifiedEventHandler(unsigned int requestId, string sender, DateTime serverUTCDateTime, FeedStatusEvent status);
  event FeedStatusModifiedEventHandler FeedStatusUpdate;

Raised when the FeedStatus of the FeedOS server change, information is sent in human readable format

4.6 Complete the connection

You have to call Disconnect() on the target Connection object to complete the connection.

5 Request

5.1 Request Id

In order to link a request with the events raised, « requestId » identifier is used within both the request and event handler methods.

All the request function prototypes are ended with the last parameter « requestId »:

```
...  uint requestId);
```

This parameter « requestId » is used as the first parameter of the event handlers:

```
...  (uint requestId, ...);
```

5.2 Request state

Four event handlers of the Connection object may be registered to be informed when a single request or a subscription is started and/or stopped.

- public delegate void RequestStartedEventHandler(uint requestId);
  public event RequestStartedEventHandler RequestStarted;
  Signal a started request.
- public delegate void RequestSucceededEventHandler(uint requestId);
  public event RequestSucceededEventHandler RequestSucceeded;
  Signal a successful request.
- public delegate void RequestAbortedEventHandler(uint requestId, uint returnCode);
  public event RequestAbortedEventHandler RequestAborted;
  Signal that the request of id *requestId* has aborted with return code *returnCode*. You can use FeedOS-Managed.API.ErrorString(returnCode) to get a comprehensive error string.
- public delegate void RequestFailedEventHandler(uint requestId, uint returnCode);
  public event RequestFailedEventHandler RequestFailed;
  Signal that the request of id *requestId* has failed with return code *returnCode*. You can use FeedOS-Managed.API.ErrorString(returnCode) to get a comprehensive error string.

6 Issueing Simple Requests

These are simple Request-Response dialogs. They are issued asynchronously (for multithreaded / parallel operations).

- public void ReferentialDumpStructure(uint requestId);
- public void ReferentialGetInstruments(List<string> instrumentCodeList, List<ushort> format, bool ignoreInvalidCodes, uint requestId);
- public void ReferentialLookup(string pattern, ushort maxHits, LookupMode Mode, List<TagNumAndValue> filter, uint requestId);
- public void ReferentialGetTradeConditionsDictionary(uint requestId);
- public void SnapshotL1(List<string> polymorphicInstrumentCodeList, List<ushort> quotationTagNumberList, bool ignoreInvalidCodes, uint requestId);

6.1 Event handlers

A subset of the Connection object event handlers are raised by a given request.

ReferentialDumpStructure

- public delegate void MarketsEventHandler(uint requestId, List<MarketCharacteristics> marketCharacteristicsList, List<MarketContent> marketContentList, DateTime lastUpdateDateTime);
  public event MarketsEventHandler MarketsHandler;

ReferentialGetInstruments

- public delegate void InstrumentsEventHandler(uint requestId, List<InstrumentCharacteristics> instrumentCharacteristicsList);
  public event InstrumentsEventHandler InstrumentsHandler;

ReferentialLookup

- public delegate void InstrumentsEventHandler(uint requestId, List<InstrumentCharacteristics> instrumentCharacteristicsList);
  public event InstrumentsEventHandler LookupInstrumentsHandler;

ReferentialGetTradeConditionsDictionary

- public delegate void GetTradeConditionsDictionaryEventHandler(uint requestId, List<TradeConditionsDictionaryEntry dictionary);
  public event GetTradeConditionsDictionaryEventHandler GetTradeConditionsDictionaryHandler;

SnapshotL1

- DEPRECATED please use SnapshotsEventHandler
  public delegate void SnapshotEventHandler(uint requestId, uint instrumentCode, InstrumentStatusL1 instrumentStatusL1);
  public event SnapshotEventHandler SnapshotHandler;

- public delegate void SnapshotsEventHandler(uint requestId, List<InstrumentStatusL1> instrumentStatusL1);
  public event SnapshotsEventHandler SnapshotsHandler;

7 Using Asynchronous Subscriptions

These are subscribe-publish dialogs.
Most of the time this is used to send realtime data but it may be used to send large volumes of information as a flow of small data chunks.
Request names finishs with "L1" or "L2" according to the level of marketdata ("L1": trades and bestquotes, "L2": orderbook refreshes).
Request names finishing with "IBar" means it involves intraday bar.

7.1 Referential subscription

- public void ReferentialDownload(List<MarketBranchId> targetBranches, List<TagNumAndValue> filteringAttributes, List<ushort> format, DateTime lastUpdateTimestamp, bool sendCreated, bool sendModified, bool sendDeleted, uint requestId);
- public void ReferentialDownloadAndSubscribe(List<MarketBranchId> targetBranches, List<TagNumAndValue> filteringAttributes, List<ushort> format, DateTime lastUpdateTimestamp, bool sendCreated, bool sendModified, bool sendDeleted, bool sendOtherData, uint requestId);

7.2 Realtime subscription

There are methods to register a list of instrument to real-time marketdata.

- public void SubscribeL1(List<string> instrumentCodeList, List<ushort> quotationTagNumberList, uint contentMask, bool ignoreInvalidCodes, uint requestId);
- public void SubscribeL2(List<string> instrumentCodeList, bool ignoreInvalidCodes, uint requestId);
- public void SubscribeMBL(List<string> instrumentCodeList, List<uint> layerIds, bool ignoreInvalidCodes, uint requestId);

There are two methods to add instrument to an existing real-time subscription using the « requestId ».

- public void SubscribeL1Add(List<string> instrumentCodeList, bool ignoreInvalidCodes, uint requestId);
- public void SubscribeL2Add(List<string> instrumentCodeList, bool ignoreInvalidCodes, uint requestId);
- public void SubscribeMBLAdd(List<string> instrumentCodeList, bool ignoreInvalidCodes, uint requestId);

There is one method to change the "ContentMask" of an existing real-time subscription using the « requestId ».

- public void SubscribeL1NewContentMask(uint contentMask, bool resendSnapshot, uint requestId);

There are two methods to remove a list of instrument from an existing real-time subscription using the « requestId ».

- public void SubscribeL1Remove(List<string> instrumentCodeList, uint requestId);
- public void SubscribeL2Remove(List<string> instrumentCodeList, uint requestId);
- public void SubscribeMBLRemove(List<string> instrumentCodeList, uint requestId);

7.3 Feed Replay

There are four methods to register a list of instrument to historical marketdata.
The subscription is done to a `source ID` provided at the same time as the FeedOS server connection parameter.
The subscription retrieves the marketdata events chronologically from the `dateTimeBegin` up to the `dateTimeEnd`.

- public void ReplayIBar(uint sourceID, DateTime dateTimeBegin, DateTime dateTimeEnd, uint requestId);
- public void ReplayL1(uint sourceID, DateTime dateTimeBegin, DateTime dateTimeEnd, uint requestId);
- public void ReplayL1L2(uint sourceID, DateTime dateTimeBegin, DateTime dateTimeEnd, uint requestId);
- public void ReplayL2(uint sourceID, DateTime dateTimeBegin, DateTime dateTimeEnd, uint requestId);
- public void ReplayFeedStatus(uint sourceID, DateTime dateTimeBegin, DateTime dateTimeEnd, uint requestId);
- public void ReplayMBO(uint sourceID, DateTime dateTimeBegin, DateTime dateTimeEnd, uint requestId);
- public void ReplayMUXED(uint sourceID, DateTime dateTimeBegin, DateTime dateTimeEnd, uint requestId);

7.4 Event handlers

ReferentialDownload/ReferentialDownloadAndSubscribe

- public delegate void BranchBeginEventHandler(uint requestId, MarketBranchId marketBranchId, uint currentQuantity, uint deletedQuantity, uint createdQuantity, uint modifiedQuantity);
  public event BranchBeginEventHandler BranchBeginHandler;
- public delegate void InstrumentsCreatedEventHandler(uint requestId, List<InstrumentCharacteristics> instrumentCharacteristicsList);
  public event InstrumentsCreatedEventHandler InstrumentsCreatedHandler;
- public delegate void InstrumentsModifiedEventHandler(uint requestId, List<InstrumentCharacteristics> instrumentCharacteristicsList);
  public event InstrumentsModifiedEventHandler InstrumentsModifiedHandler;

- public delegate void InstrumentsDeletedEventHandler(uint requestId, List<uint> deletedInstrumentsCodeList);
  public event InstrumentsDeletedEventHandler InstrumentsDeletedHandler;

- public delegate void MarkerTimestampEventHandler(uint requestId, DateTime markerTimestamp);
  public event MarkerTimestampEventHandler MarkerTimestampHandler;

- public delegate void RealtimeBeginEventHandler(uint requestId);
  public event RealtimeBeginEventHandler RealtimeBeginHandler;

Level 1

- DEPRECATED please use SnapshotsEventHandler
  public delegate void SnapshotEventHandler(uint requestId, uint instrumentCode, InstrumentStatusL1 instrumentStatusL1);
  public event SnapshotEventHandler SnapshotHandler;

- public delegate void SnapshotsEventHandler(uint requestId, List<InstrumentStatusL1> instrumentStatusL1);
  public event SnapshotsEventHandler SnapshotsHandler;

- public delegate void TradeEventExtEventHandler(uint requestId, uint instrumentCode, DateTime serverUTCDateTime, DateTime marketUTCDateTime, QuotationTradeEventExt quotationTradeEventExt);
  public event TradeEventExtEventHandler TradeEventExtHandler;

- public delegate void ValuesUpdateEventHandler(uint requestId, uint instrumentCode, DateTime marketUTCDateTime, QuotationValuesUpdate quotationValuesUpdate);
  public event ValuesUpdateEventHandler ValuesUpdateHandler;

- public delegate void MarketNewsEventHandler(uint requestId, MarketNews marketNews);
  public event MarketNewsEventHandler MarketNewsHandler;

- public delegate void MarketStatusEventHandler(uint requestId, MarketStatus marketStatus);
  public event MarketStatusEventHandler MarketStatusHandler;

Deprecated Level 2

- public delegate void OrderBookDeltaRefreshEventHandler(uint requestId, uint instrumentCode, ValueType serverUTCDateTime, OrderBookDeltaRefresh orderBookDeltaRefresh);
  public event OrderBookDeltaRefreshEventHandler OrderBookDeltaRefreshHandler;

- public delegate void OrderBookMaxVisibleDepthEventHandler(uint requestId, uint instrumentCode, byte maxVisibleDepth);
  public event OrderBookMaxVisibleDepthEventHandler OrderBookMaxVisibleDepthHandler;

- public delegate void OrderBookRefreshEventHandler(uint requestId, uint instrumentCode, DateTime serverUTCDateTime, OrderBookRefresh orderBookRefresh);
  public event OrderBookRefreshEventHandler OrderBookRefreshHandler;

- DEPRECATED please use OrderBookSnapshotsEventHandler
  public delegate void OrderBookSnapshotEventHandler(uint requestId, uint instrumentCode, OrderBook orderBook);
  public event OrderBookSnapshotEventHandler OrderBookSnapshotHandler;

- public delegate void OrderBookSnapshotsEventHandler(uint requestId, List<InstrumentStatusL2> instrumentStatusL2);
  public event OrderBookSnapshotsEventHandler OrderBookSnapshotsHandler;

Level 2 (Extended MBL)

- public delegate void MBLAddedInstrumentsEventHandler(uint requestId, List<uint> FOSInstrumentCodeList);
  public event MBLAddedInstrumentsEventHandler MBLAddedInstrumentsHandler;
- public delegate void MBLFullRefreshEventHandler(uint requestId, List<MBLSnapshot> snapshots);
  public event MBLFullRefreshEventHandler MBLFullRefreshHandler;
- public delegate void MBLOverlapRefreshEventHandler(uint requestId, MBLOverlapRefresh refresh);
  public event MBLOverlapRefreshEventHandler MBLOverlapRefreshHandler;
- public delegate void MBLDeltaRefreshEventHandler(uint requestId, MBLDeltaRefresh delta);
  public event MBLDeltaRefreshEventHandler MBLDeltaRefreshHandler;
- public delegate void MBLMaxVisibleDepthEventHandler(uint requestId, MBLMaxVisibleDepth maxVisibleDepth);
  public event MBLMaxVisibleDepthEventHandler MBLMaxVisibleDepthHandler;

MBO

- public delegate void MarketSheetSnapshotEventHandler(uint requestId, uint instrumentCode, DateTime lastServerUTCDateTime, DateTime lastMarketUTCDateTime, List<MarketSheet> marketSheet);
  public event MarketSheetSnapshotEventHandler MarketSheetSnapshotHandler;
- public delegate void NewOrderEventHandler(uint requestId, uint instrumentCode, DateTime serverUTCDateTime, DateTime marketUTCDateTime, List<TagNumAndValue> context,uchar inSide, MarketSheetEntry marketSheetEntry, ushort inLevel);
  public event NewOrderEventHandler NewOrderHandler;
- public delegate void ModifyOrderEventHandler(uint requestId, uint instrumentCode, DateTime serverUTCDateTime, DateTime marketUTCDateTime, List<TagNumAndValue> context,uchar inSide, MarketSheetEntry marketSheetEntry, ushort inOldLevel, ushort inNewLevel);
  public event ModifyOrderEventHandler ModifyOrderHandler;
- public delegate void RemoveOneOrderEventHandler(uint requestId, uint instrumentCode, DateTime serverUTCDateTime, DateTime marketUTCDateTime, List<TagNumAndValue> context,uchar inSide, string orderID, ushort inLevel);
  public event RemoveOneOrderEventHandler RemoveOneOrderHandler;
- public delegate void RemoveAllPreviousOrdersEventHandler(uint requestId, uint instrumentCode, DateTime serverUTCDateTime, DateTime marketUTCDateTime, List<TagNumAndValue> context,uchar inSide, string orderID, ushort inLevel);
  public event RemoveAllPreviousOrdersEventHandler RemoveAllPreviousOrdersHandler;
- public delegate void RemoveAllOrdersEventHandler(uint requestId, uint instrumentCode, DateTime serverUTCDateTime, DateTime marketUTCDateTime, List<TagNumAndValue> context,uchar inSide);
  public event RemoveAllOrdersEventHandler RemoveAllOrdersHandler;

- public delegate void RetransmissionEventHandler(uint requestId, uint instrumentCode, DateTime serverUTCDateTime, DateTime marketUTCDateTime, MarketSheet> marketSheet);
  public event RetransmissionEventHandler RetransmissionHandler;

- public delegate void ValuesUpdateOneInstrumentEventHandler(uint requestId, uint instrumentCode, DateTime serverUTCDateTime, DateTime marketUTCDateTime, List<TagNumAndValue> values);
  public event ValuesUpdateOneInstrumentEventHandler ValuesUpdateOneInstrumentHandler;

Intraday Bar

- public delegate void IntradayBarsEventHandler(uint requestId, List<QuotationIntradayBar> quotationIntradayBar);
  public event IntradayBarsEventHandler IntradayBarsHandler;

7.5 Stopping the subscription

Of course a user can decide to terminate an ongoing subscription. This is done by calling a « stop » method which is the counterpart of the « start » method described above.

This method always succeeds. No callback is triggered.

No more event handlers will be triggered for this subscription (although one could still be running in a different thread) so you can unregister the functions. You may want to implement some synchronization to enforce that a possibly running callback terminate before you can assume this subscription is effectively « off ».

8 Reference Guide

Hereunder, we present an exhaustive list of Types, simple requests and subscriptions available.

8.1 Types

8.1.1 FeedState enum

```
public enum FeedState
{
  FeedState_Active /** normal with activity */,
  FeedState_ProbablyNormal /** looks like normal */,
  FeedState_ProbablyDisrupted /** looks like disrupted */,
  FeedState_Disrupted_TechnicalLevel /** not usable (technical reasons) */,
  FeedState_Disrupted_ExchangeLevel /** not usable (N/A at exchange level) */,
}
```

8.1.2 FeedDescription class

Description of a FeedStatus or FeedStatusEvent, embedding the name of the source and it's internal ids

```
public class FeedDescription
{
  public FeedDescription(String feedName, List<ushort> internalSourceIds) {}

  #region Properties
  public String FeedName
  {
    get { return m_FeedName; }
  }
  public List<ushort> InternalSourceIds
  {
    get { return m_InternalSourceIds; }
  }
  #endregion


  #region Members
  protected String m_FeedName;
  protected List<ushort> m_InternalSourceIds;
  #endregion
}
```

8.1.3 FeedUsability class

FeedUsability of a FeedStatus, embedding a FeedState describing a source state. Additional flags depicts the risk attached to the state

```
public class FeedUsability
{
  public FeedUsability(FeedState feedState, bool latencyPenalty, bool
      outOfDateValues, bool badDataQuality) {}
```

```
  #region Properties
  public FeedState FeedState
  {
    get { return m_FeedState; }
  }
  public bool LatencyPenalty
  {
    get { return m_LatencyPenalty; }
  }
  public bool OutOfDateValues
  {
    get { return m_OutOfDateValues; }
  }
  public bool BadDataQuality
  {
    get { return m_BadDataQuality; }
  }
  #endregion

  #region Members
  protected FeedState m_FeedState;
  protected bool m_LatencyPenalty;
  protected bool m_OutOfDateValues;
  protected bool m_BadDataQuality;
  #endregion
}
```

8.1.4 FeedServiceStatus class

FeedServiceStatus is a FeedUsability related to a service, it embeds the service name

```
public class FeedServiceStatus
{
   public FeedServiceStatus(String serviceName, FeedUsability usability) {}

   #region Properties
   public String ServiceName
   {
     get { return m_ServiceName; }
   }
   public FeedUsability Usability
   {
     get { return m_Usability; }
   }
   #endregion

   #region Members
   protected String         m_ServiceName;
   protected FeedUsability m_Usability;
   #endregion
}
```

8.1.5 FeedStatus class

FeedStatus composed of a FeedDescription, an overall FeedUsability and a List of FeedServiceStatus.

```
public class FeedStatus
{
   public FeedStatus(FeedDescription feedDesc, FeedUsability overallUsability,
       List<FeedServiceStatus> serviceStatus){}

   #region Properties
   public FeedDescription FeedDescription
   {
      get { return m_FeedDescription; }
   }
   public FeedUsability FeedUsability
   {
      get { return m_OverallUsability; }
   }
   public List<FeedServiceStatus> ServiceStatus
   {
      get { return m_ServiceStatus; }
   }
   #endregion

   #region Members
   protected FeedDescription m_FeedDescription;
   protected FeedUsability m_OverallUsability;
   protected List<FeedServiceStatus> m_ServiceStatus;
   #endregion
}
```

8.1.6 FeedStatusEvent class

A FeedStatusEvent is the equivalent of a FeedStatus, but all informations are put into a String Format, more for a debug/trace behavior, as no behavior can be triggered on a particular FeedState modification

```
        public class FeedStatusEvent
        {
            public FeedStatusEvent(FeedDescription feedDesc, String eventType,
                List<String> eventDetails) {}

            #region Properties
            public FeedDescription FeedDescription
            {
                get { return m_FeedDescription; }
            }
            public String EventType
            {
                get { return m_EventType; }
            }
```

```
            public List<String> EventDetails
            {
                get { return m_EventDetails; }
            }
            #endregion

            #region Members
            protected FeedDescription m_FeedDescription;
            protected String m_EventType;
            protected List<String> m_EventDetails;
            #endregion
        }
```

8.2 Service CONNECTION

8.2.1 CONNECTION Simple Requests

Connect Connecting to a FeedOS server.

Disconnect Disconnecting from a FeedOS server.

8.2.2 CONNECTION Subscriptions

SubscribeToFeedStatus Launching subscription to any modification of the FeedStatus, reflecting Quality of Service informations and market data source availability.

8.3 Service REFERENTIAL

8.3.1 REFERENTIAL Simple Requests

ReferentialDumpStructure Retrieve the « structure » of the referential data. This is made of a list of markets, a list of « branches », and number of known instruments. A branch is a kind of containter for instruments of a given type (tags CFICode and SecurityType) in a given market (tag FOSMarketId). Arguments: none.

ReferentialGetInstruments Retrieve the characteristics of instruments. Arguments: a list of instrument codes + an optional list of tags to retrieve.

ReferentialLookup Dynamically search for instruments. Arguments: a pattern + optional filter and an optional list of tags to retrieve.

ReferentialGetTradeConditionsDictionary Retrieve the trade conditions dictionary where trade conditions flags combination are identified by a key.

8.3.2 REFERENTIAL Subscription-like

ReferentialDownload Download whole or subset of a referential. Arguments: list of referential branches, attributes filters, output format, boolean to get creation, modified and deleted instruments

8.3.3 REFERENTIAL Subscription

ReferentialDownloadAndSubscribe Download whole or subset of a referential, and subscribe to referential data creation/modification and deletion. Arguments: list of referential branches, attributes filters, output format, boolean to get creation/modified/deleted instruments and boolean to get other data.

8.4 Service QUOTATION

8.4.1 QUOTATION Simple Requests

SnapshotL1 Retrieve the « Level 1 » current values for one or several instruments. Arguments: list of instrument codes, a list of tags to retrieved.

GetHistoDaily Retrieve the Open/Close/High/Low/CumulatedVolume/TotalAssetTraded daily values for a given instrument, over a specified period of time. Arguments: instrument code, begin date + end date

GetHistoIntraday DEPRECATED - use GetHistoIntraday2 - Retrieve the trade values for a given instrument, over a specified period of time. Arguments: instrument code, begin timestamp + end timestamp + number of « points » to retrieve (0 means « all », else a sampling is done).

GetHistoIntraday2 Retrieve the trade values for a given instrument, over a specified period of time. Arguments: instrument code, begin timestamp, end timestamp, number of « points » to retrieve (0 means « all », else a sampling is done).

8.4.2 QUOTATION Subscriptions

SubscribeL1 Subscribe to « Level 1 » real-time data for one or several instruments. Arguments: list of instrument codes, « content mask » to filter events.

SubscribeL2 DEPRECATED - use SubscribeMBL - Subscribe to « Level 2 » real-time data for one or several instruments. Arguments: list of instrument codes, boolean to ignore or not invalid instrument code (if false an unknown code will make the whole subscription fail)

SubscribeMBL Subscribe to « Extended MBL » real-time data for one or several instruments. Arguments: list of instrument codes, boolean to ignore or not invalid instrument code (if false an unknown code will make the whole subscription fail).

SubscribeMBO Subscribe to « Market By Order » real-time data for one instrument. Arguments: instrument code.

ReplayL1 Subscribe to « Level 1 » historical data to a given source. Arguments: source ID, beginning and end of period.

ReplayL2 Subscribe to « Level 2 » historical data to a given source. Arguments: source ID, beginning and end of period.

ReplayL1L2 Subscribe to both « Level 1 » and « Level 2 » historical datum to a given source. Arguments: source ID, beginning and end of period.

ReplayMBO Subscribe to MBO (Market By Order) historical data to a given source. Arguments: source ID, beginning and end of period.

ReplayFeedStatus Subscribe to FeedStatus historical data to a given source. Arguments: source ID, beginning and end of period.

ReplayMUXED Subscribe to all historical data (L1,L2,MBO,FeedStatus) to a given source. Arguments: source ID, beginning and end of period.

8.4.3 QUOTATION Requests, related to subscriptions

SubscribeL1Add Modify an existing « Level 1 » real-time data subscription adding one or several instruments. Arguments: list of instrument codes, boolean to ignore or not invalid instrument code (if false an unknown code will make the whole subscription fail).

SubscribeL1Remove Modify an existing « Level 1 » real-time data subscription removing one or several instruments. Arguments: list of instrument codes.

SubscribeL2Add DEPRECATED - use SubscribeMBLAdd - Modify an existing « Level 2 » real-time data subscription adding one or several instruments. Arguments: list of instrument codes, boolean to ignore or not invalid instrument code (if false an unknown code will make the whole subscription fail).

SubscribeL2Remove DEPRECATED - use SubscribeMBLRemove - Modify an existing « Level 2 » real-time data subscription removing one or several instruments. Arguments: list of instrument codes.

SubscribeMBLAdd Modify an existing « Extended MBL » real-time data subscription adding one or several instruments. Arguments: list of instrument codes, boolean to ignore or not invalid instrument code (if false an unknown code will make the whole subscription fail).

SubscribeMBLRemove Modify an existing « Extended MBL » real-time data subscription removing one or several instruments. Arguments: list of instrument codes.