CONTACT

Asia Pacific
+852.2846.3500

Canada
+1.416.874.0900

EMEA
+44.20.7670.4000

United States
+1.212.588.4000

info@itg.com
www.itg.com

POSIT MTF Market data technical specification

ITG Multicast API

Version 1.1

NOTICE & DISCLAIMER

These materials are for informational purposes only, and are not intended to be used for trading or investment purposes or as an offer to sell or the solicitation of an offer to buy any security or financial product. The information contained herein has been taken from trade and statistical services and other sources we deem reliable but we do not represent that such information is accurate or complete and it should not be relied upon as such. No guarantee or warranty is made as to the reasonableness of the assumptions or the accuracy of the models or market data used by ITG or the actual results that may be achieved. These materials do not provide any form of advice (investment, tax or legal). ITG Inc. is not a registered investment adviser and does not provide investment advice or recommendations to buy or sell securities, to hire any investment adviser or to pursue any investment or trading strategy. All functionality described herein is subject to change without notice. These materials are highly confidential and are not to be copied, displayed or transmitted in any form without the prior written permission of ITG.

Broker-dealer products and services are offered by: in the U.S., ITG Inc., member FINRA, SIPC; in Canada, ITG Canada Corp., member Canadian Investor Protection Fund ("CIPF") and Investment Industry Regulatory Organization of Canada ("IIROC"); in Europe, Investment Technology Group Limited, registered in Ireland No. 283940 ("ITGL") (the registered office of ITGL is Block A, Georges Quay, Dublin 2, Ireland). ITGL is authorized and regulated by the Central Bank of Ireland; in Asia, ITG Hong Kong Limited (SFC License No. AHD810), ITG Singapore Pte Limited (CMS License No. 100138-1), and ITG Australia Limited (AFS License No. 219582). All of the above entities are subsidiaries of Investment Technology Group, Inc. MATCHNow® is a product offering of TriAct Canada Marketplace LP ("TriAct"), member CIPF and IIROC. TriAct is a wholly owned subsidiary of ITG Canada Corp.

Table of Contents

1. Overview

Pre-trade market data from the POSIT MTF Auction segment may be accessed via ITG's market data feed known as the "ITG Multicast API".

For the POSIT MTF Auction segment, indicative price and volume information is disseminated during an auction. Please refer to POSIT MTF rulebook for more information.

For access to the ITG Multicast API, contact POSIT Support via Email: ITGE-ElectronicSales@itg.com or Phone: +44 20 7670 4066

A username, password, connection details and API toolkit for the ITG Multicast API will be provided on request.

2. Architecture

The ITG URT Multicast API is a C/C++ messaging API based upon ITG proprietary reliable multicast protocol.

The API is event based and provides two types of callbacks:

- **Feed** callbacks are used to deliver price events and publisher events.
    - Price events are triggered as prices are decoded and pulled out of the queue.
    - Publisher events are triggered during state changes, such as a publisher going up or down or if data is dropped.
- **Log** callbacks may be overridden to output log events to your own custom mechanism, which is always recommended. By doing so, API log messages would be saved into the same log file of your application, which helps to troubleshoot problems

Client code should return all callbacks as soon as possible. Failure to do may cause inbound packet gapping, which leads to bad data quality and higher latency. Missing or dropped data is referred to as a **gap**.

The API will attempt to recover from data loss by sending resend requests to the publisher. The publisher maintains a ring-queue of sent messages. If the resent request is beyond the range of the messages cached in the ring-queue then data is considered unrecoverable, and referred to as a **true gap**. True gaps in data may cause dropped prices.

Upon initialization, the API will submit a request for a snapshot of the current market. Snapshot data may arrive out of order so we recommend that you do not start acting on price data until after the snapshot has completed.

Please review the C++/PAClient, sample in the API toolkit for examples of how to use the API.

3. Initialization

3.1 Feed Initialization (MdFeedInit)

A MdFeedInit structure defines the options that you wish to configure for a given feed. Many MdFeedInit objects may be associated with any given MdClient object.

Field Name	Description	Configuration Optional(O)/ Mandatory(M)/ NotApplicable(NA)
feedId	The feed that you want to process events for. For a list of feedId's, please refer to Appendix A	M
pEvents	An event sink derived off of MdEvents to use. It must be valid for the lifetime of the MdClient class. Refer to section 7.1 for a description of the MdEvents callback type.	M
uiHeartbeatDC	The number of seconds to wait until disconnecting due to a lack of heartbeats. A heartbeat is sent out once every second, so if you set this to 10 then you will disconnect after 10 seconds of not receiving heartbeats.	O
uiHeartbeatSuspect	The number of seconds to wait until calling OnFeedSuspect() due to a lack of heartbeats.	O
uiQueueId	The feed queue id. May range from 0 to 100. Multiple feeds can share a queue or you can have a unique queue per feed. Leave as 0 to use the default behaviour which is 1 queue for every available CPU minus the number of IO pools. There is one thread per queue. MdEvent callbacks will take place on the queue's thread.	O
uiIoPoolId	The feed IO thread pool id. May be any unsigned integer. Multiple feeds can share a IO thread pool or you can have a unique pool per feed. Leave as 0 to use the default behaviour which is 1 pool for every three available cpus. There is one thread per IO thread pool.	O
bSendResendRequests	Send resend requests for dropped data. This is set to true by default.	O
uiQuoteSizeMultiplier	A configurable parameter to adjust inbound quote sizes by a certain factor.	O
bFilterDepthWithMpid	A configurable parameter to filter depth based on mpid or userdata. This is set to false by default.	O
arrExchFilters	A configurable parameter to prepare/maintain a list of exchange codes to be filtered out.	O
uiNumExchFilters	Holds the number of exchange codes filtered out in above list.	O
arrExchTradeFilters	A configurable parameter to prepare/maintain a list of trade exchange codes to be filtered out.	O
uiNumExchTradeFilters	Holds the number of trade exchange codes filtered out in above list.	O
arrExchReplace	A configurable parameter to prepare/maintain a list of exchange codes to be replaced.	O
uiNumExchReplace	Holds the number of exchange codes replaced in above list.	O
uiFromSuffixIndex	The equity symbol type that you want to convert from using symbol converter. For a list of equity symbol types, please refer to mdpricestructs.h	O
uiToSuffixIndex	The equity symbol type that you want to convert to using symbol converter. For a list of equity symbol types, please refer to mdpricestructs.h	O

3.2 MdClient Initialization (MdInit)

Use the MdInit structure to initialize an MdClient object with any number of feeds.

Field Name	Description	Configuration Optional(O)/ Mandatory(M)/ NotApplicable(NA)
pFeedInit	Array of MdFeedInit structures.	Refer to table 3.1
szUserName	A configurable username to be authenticated for receiving multicast data from SMD's. Refer to Appendix A for values	M
szPassword	A configurable password mapped to above user. Password should be identical to username	M
uiNumFeeds	Number of MdFeedInit structures present.	M
pDiscoveryAddress[1]	Array of discovery MdAddress structures used to detect the location of a given feed. Each feed will publish out a heartbeat message containing multicast feed address details every second. The MdClient uses the heartbeat message to determine what multicast address it should listen to for feed updates. Refer to Appendix A for values	M
uiNumDiscoveryInterfaces	Number of discovery interfaces to listen to, e.g. for multiple VLANs.	M
uiDiscoveryQueueId	The discovery queue id. Leave as 0 to use the default queue.	O
uiDiscoveryIoPoolId	The discovery IO thread pool id. Leave as 0 to use the default IoPool.	O
pLogSink	Log sink to use if you wish to override the default logging sink or NULL if you do not. If NULL, printf() is used for all logging output, otherwise output is handled by your object which must be valid for the lifetime of the MdClient class. We recommend always override with your log object so MdClient log message could be saved along with your logging mechanism.	O
pQueueEvents	Queue event sink. If NULL, no queue events will be processed, otherwise events will be processed by the object which must be valid for the lifetime of the MdClient class.	O
usResendPortRangeBegin	Start UDP point to point port range. The client may bind to any port in the port range to listen for resent price data. The default range is from 34000 to 34200.	O
usResentPortRangeEnd	End UDP point to point port range.	O
pLatencyInit	Configurable parameter to read and provide MD latency monitor setup.	O
szDiscoveryLocation	This is a configurable value to read/publish data on a particular multicast address/port which is further bounded by a location specific string.	O
uiDiscoveryTimeout	This is a configurable value to allow the application to wait for a pre-defined duration from the point of receiving the first discovery message untill it begins processing successive discovery messages. This is defaulted to 5000.	O
uiIoType	The OS specific way to choose the way we consume and send data. For example, on windows you can use the following: 0. Default → Choose for me 1. GetQueuedCompletionStatus → Allows to dequeue 1 IO at a time 2. GetQueuedCompletionStatusEx → Allows to dequeue many IO's at a time	O
uiIoWaitMs	Time in milliseconds the IO thread will wait for IO on a socket before timing out and yielding back to the IO thread.	O

	OS specific and may be optional depending on uiType	
uiIoNumReads	This is an optional configurable value which represents the total number of interfaces to decode call in OnReadData(). This is defaulted to 1.	O
uiSecondsUntilFailure	This is a configurable value which represents a timeout until the feed is marked as a suspect. This is defaulted to 30 seconds. For example a check is run every second. if no data is received for 30 seconds then the feed is marked as a suspect.	O
btsStartTime	Start time to run above check(s).	O
btsEndTime	End time to run above check(s).	O
pMdFeedPolicy	A pointer to a policy object which is used for processing and publishing outbound trades and other mdprice messages.	O
pIoThreadCpuAffinities	Set which CPU's thread is allowed to run on (OS specific) and may be not used by all OS's	O
uiNumIoThreadCpuAffinities	Holds the total number of Io pool thread CPU affinities configured.	O
pWorkerThreadCpuAffinities	Set which CPU's worker thread is allowed to run on (OS specific) and may be not be used by all OS's	O
uiNumWorkerThreadCpuAffinities	Holds the total number of workerl thread CPU affinities configured.	O
pDefinitionsInit	Hold the pointer pointing to MdDefinitions Init which holds the configured currency code file "urt_currency_codes.csv". This file can be found under folder "docs" in mdclient distribution package. If NULL, environment variable "URTDIR_DEFINITIONS" must be present and contain the valid path to a local folder that stores the same file.	M
pStatisticsTimerAffinity	Holds the pointer pointing to MdClient Statistics Timer Affinity mask.	O

4. Sequencing

The API uses lazy sequencing when processing price updates. In stricter API's, you would often need to wait for a resend request to fill in data gaps prior to processing. Of which, the major drawback is significant added latency. It could take anywhere from a few milliseconds to a few hundred milliseconds to recover from gapping.

However, lazy sequencing allows the end user to decide if they wish to wait for a resend to complete before processing new data, which is essential for latency sensitive algorithms because it allows the algorithm to access new data while data gaps are being recovered.

4.1 Trade sequencing

Trade messages have a per contract type sequence. If a trade arrives with a higher than expected sequence it will be processed immediately. If a trade arrives with a lower than expected sequence on the same contract it will be thrown away.

For example:

sequence 1, B7T7721 trade
OnTrade () for B7T7721 is called.
sequence 2, 0263494 trade (the message is dropped)
sequence 3, 0263494 trade
OnTrade () for 0263494, outOfOrder = true
replay of sequence 2 arrives. OnTrade() is not called because the current trade sequence for 0263494is 3.

5. Snapshots

A snapshot of the market will be sent to you when you initially connect to a feed.

Trade feed snapshots contain price record messages. No special processing is required to process Trade feed snapshots, you may begin using Trade feed data immediately.

5.1 Snapshot_status

Field Name	Description
snapshot_status_failed	Updated when snapshot for a given feed fails.
snapshot_status_started	Updated when snapshot for a given feed is started.
snapshot_status_complete	Updated when snapshot for a given feed is complete.
snapshot_status_pending	Updated when snapshot for a given feed is pending.
snapshot_status_rejected	Updated when snapshot for a given feed is rejected owing to user authentication failure.

6. Callbacks

6.1 Feed Callbacks (MdEvents)

Refer to *Appendix A* for example message work flow for POSIT Auction indicative price and volume updates.

Callback Name	Description
OnTrade	A trade message is ready to be processed.
OnPriceRecord	A price record message is ready to be processed.
OnImbalance	An imbalance message is ready to be processed.
OnUnknownMessage	An unknown message was detected. This may happen if you are on an older version of the API that does not support a new message type.
OnFeedStatus	The status of the feed.
OnFeedConnected	The feed is connected and the snapshot will begin.
OnFeedDisconnected	Called if the number of heartbeats set in uiHeartbeatDC is missed.
OnFeedSnapshotStatus	Called when the snapshot first starts, then called again once the snapshot is complete or if the snapshot has failed.
OnFeedInSequence	Called when the feed has recovered missing data and it is back in sequence.
OnFeedOutOfSequence	Called when a drop-in data has been detected.
OnCleanupInfoUserData	Called on shutdown when all instrument info records are being removed. You can cleanup your user data here.
OnCleanupQuoteTradeUserData	Called on shutdown when all quote/trade records are being removed.

6.2 Log Callbacks (MdLogSink)

Callback Name	Description
OnLogMessage	Implement this callback and log the given message with your own logging mechanism. This ensures mdclient log stores along with client application's log for easy support and troubleshooting.

7. Message structures

The following message structures are located in mdpricestructs.h. The structure name is the name in parenthesis from each section header. For example, in section header 7.1 Message header (msg_hdr), msg_hdr is the structure name.

7.1 Message header (msg_hdr)

A message header is present on price events.

Field Name	Description	Example
sequence	The sequence number of the item. Each sent multicast packet will step up the sequence number by 1. The sequence number will wrap around to 0 if the sequence goes past 4,294,967,295. More than one message may be in the same packet and share the same sequence number.	1234
feedId	The feed that originated the message. For a list of feedId's, please refer to Appendix A	289
feedInstance	The unique id of the feed. If the feed is restarted or if a backup feed takes over from the primary, this id will change after you will receive the OnFeedDisconnected() and OnFeedConnected() events.	12(Not applicable)
msgStatus	The origin of the message. This can be: message_status_realtime = real time price data. message_status_snapshot = data from a startup snapshot. message_status_replay = replay of dropped data from resend request.	0 (message_status_realtime)
outOfOrder	Set to true if the message is processed out of order.	False(Not applicable)
lastInGroup	Set to true if this is the last message of current UDP packet.	true
feedTime	Time (UTC) from POSIT.	tv_sec: 1501683296 tv_usec: 20895
processTime	Time (UTC) that ITG URT normalizes message.	tv_sec: 1501683297 tv_usec: 30895
recvTime	Time (UTC) that this message was received by the API.	tv_sec: 1501683298 tv_usec: 40895
exchangeSequence	The sequence of the message as set by the exchange. If 0, then exchange did not supply a sequence or a sequence was unavailable.	2345
userData	User set user data from MdClient::SetQuoteTradeUserData(),	NULL(Not applicable)
flags	State flags for the message. This is a bitmap with the following possible non-overlapping values: msg_flag_none = none of the following cases applies msg_flag_has_blob = the message contains a blob (below) msg_flag_clear_book = the message was generated internally to indicate the book is being cleared, typically because the upstream feed was lost	0 (msg_flag_none)
biDynamic	A blob containing any other information for this update that could not be contained in the body of the message.	NULL(Not applicable)

7.2 Instrument Information (instrument_info)

Most price feed messages contains an instrument_info. The combination of symbol, countryCode, exchangeCode and currencyCode identify a unique instrument.

Field Name	Description	Example
symbol	SEDOL.	B7T7721
countryCode[2]	ITG internal Country Code	3 (GBR)
exchangeCode	Integer value of 51	51
currencyCode[3]	The currency in which the update is quoted.	3 (GBP)
contractId	Contract id for the given instrument. This id is not unique and may change intraday.	123456(Not applicable)
userData	User set user data from MdClient::SetInfoUserData().	Not applicable

7.3 Last Price (last_price)

Field Name	Description	Example
price	Integer encoded price.	99000000
size	Size at the given price.	100
exchange	Exchange that the price originated from.	51 (equity_exchange_code_posit)
indicator	Special handling code for the traded price. For example, the trade could be out of sequence or an official close	114 (trade_indicator_continuous_auction)
secondaryIndicators	Additional trade indicators.	0 (trade_indicator_regular)
time	Time (UTC) that the price was disseminated from the exchange	tv_sec: 1501683296 tv_usec: 20895

7.4 Trade (trade)

Field Name	Description	Example
Msg_hdr	Inherits from.	See Message header
Info	The instrument data.	See Instrument Information
priceType[4]	The price type used to decode the price from a long to a double.	16 (price_type_decimal_6)
Price	Last Price object containing trade price and size details.	See Last Price
volume	Total shares of all trades happened so far	Not applicable
compositeVolume	Total traded shares across all exchanges. Not applicable to Periodic Auction.	Not applicable
contextFlags	Bitwise flags. Can be multiple context_flag items Or'd together.	0 (trade_context_flag_normal)

7.5 Price Record (price_record)

Field Name	Description	Example
Msg_hdr	Inherits from.	See Message header
Info	The instrument data.	See Instrument Information
priceType4	The price type used to decode the price from a long to a double.	16 (price_type_decimal_6)
Ask	Not applicable to Periodic Auction	Not applicable

[2] use CountryCodeToName() to convert to ascii string

[3] use CurrencyCodeToName() to convert to ascii string

Bid	Not applicable to Periodic Auction	Not applicable
Last	Last Price structure containing last price details.	See Last Price
volume	Not applicable to Periodic Auction	Not applicable
Ohlc	Not applicable to Periodic Auction	Not applicable

7.6 Imbalance (imbalance)

Field Name	Description	Example
Msg_hdr	Inherits from.	See Message header
info	The instrument data that apply to the imbalance.	See Instrument Information
crossType	The type of imbalance cross_type_none: periodic auction off cross_type_market_imbalance: periodic auction on other cross_types: unused	4 (cross_type_market_imbalance)
shares	Not applicable to Periodic Auction	Not applicable
side	The side of the imbalance. Always imbalance_direction_indeterminate	2 (imbalance_direction_indeterminate)
pairedShares	Indicative size.	100
priceType4	The price type used for the indicative prices.	16 (price_type_decimal_6)
currentReferencePrice	Indicative price.	99000000

8. Decoding a price

You can use the feedQuote2double() function to decode a pricetype and price to a double.

```
double d = feedQuote2double( trade.priceType, trade.last.price );
```

9. Counters

9.1 Queue counters (MdQueueCounters)

Counter Name	Description
uiItemsDropped	Total number of items dropped due to the queue filling up.
uiQueueSize	The current queue size.
uiMaxQueueSizeHit	The largest queue size hit so far.
timeMaxQueueSizeHit	The time that the queue hit that size.
uiQueueMaxSize	The current queue max size.
uiItemsProcessed	The number of queue items processed.

9.2 Feed counters (MdFeedCounters)

Counter Name	Description
uiGaps	Incremented whenever a gap is detected, and a replay can be done to fill in the data.
uiGapsTotal	When uiGaps is incremented, the sequence difference is added to this counter. It represents the total number of missed messages

	throughout the life of the process which could be recovered.
uiTrueGaps	Incremented whenever a gap is detected, and a replay cannot be done to fill in the data.
uiTrueGapsTotal	When uiTrueGaps is incremented, the sequence difference is added to this counter. It represents the total number of missed messages throughout the life of the process which could not be recovered.
uiResentRequests	The number of resend requests sent so far.
uiOutofSequence	The number of times we went out of sequence. This is not the same as gaps. There can be multiple gaps when we are out of sequence until we get back in sequence.
uiResequenceGiveup	The number of times we failed to get back in sequence due to the remote size wrapping around its cache of the last X number of messages.
uiArrayMisses	If a miss on the array occurs, a slower hash lookup will occur instead.
uiArrayHits	Array lookup hits on the contract id.
uiArrayMismatch	If a mismatch on the array occurs, a slower hash lookup will occur instead.
uiMsgsProcessed	Total number of messages processed.
uiMsgsDiscarded	Total number of messages discarded.
uiMsgsDiscardedNotConnected	Total number of messages discarded owing to packet status disconnect.
uiMsgsDiscardedInvalidFeedId	Total number of messages discarded owing to invalid Feed Id.
uiMsgsDiscardedOldPacket	Total number of messages discarded owing to processing of old message.
uiMsgsDiscardedOldMsg	Total number of messages discarded owing to processing of old out of sequence message i.e., could be an older trade messages.
uiResendsProcessed	Total number of replay messages processed.
uiNumPackets	Total number of packets processed.
uiUnknownMessages	Count of unknown messages. If this increments then you may need to upgrade your API.
uiSuspect	The number of times the feed was marked suspect.
uiDisconnects	The number of times the feed was disconnected.
uiInstanceChanges	The number of times the publishing data source was restarted and/or the number of times we fell over to a backup source.
uiHashItems	The number of hash items.
uiHashCollisions	The number of hash collisions.
uiMaxHashDepth	The max hash bucket depth.
feedAddr	The current multicast feed address.
uiIoQueued	The number of pending feed items on the queue.
uiReplayQueueSize	The replay ring queue size of the feed publisher.
uiOutstandingDeletes	Total number of outstanding deletes which can be removed later if we never recover the add/update that came before this delete.
szLocation	Discovery location bound to the corresponding discovery address/port used for reading/publishing multicast data.
tickLastUpdate	Holds the last updated tick when the application starts up or ticks received for processing.
uiLastSequence	Holds the last updated sequence in msg_hdr::sequence.

10. Operating System Dependencies

10.1 Windows

The Windows VS2013 redistributable must be installed. For your convenience, it is included in the “redist” folder of the API toolkit.

64-bit and 32-bit C++ Windows libraries are available.

Windows libraries are built using VS2013 Update 3.

Either utils.dll and mdclient.dll or utils64.dll and mdclient64.dll must be in your path or working directory when using the C++ API.

10.2 Linux

64-bit and dynamic Linux binaries are available. The libraries depend on the following packages: glibc, libstdc++, libgcc, and kernel-devel.

Dynamic libraries only (follow these instructions if you are not sure where to place the dynamic libraries):

1. Place the shared object files in /usr/local/lib
2. Either create or edit /etc/ld.so.conf to contain the path /usr/local/lib
3. As root run /sbin/ldconfig
4. You can now link with the shared objects by passing the -l flag followed by the library name to your compiler (eg. -lmdclient32, -lutils32, etc.)

Linux libraries are built using GCC version 6.2.0 on RedHat 6.4 64 bit

11. Common Log Messages

Requesting replay on feed <feed_id> from <start sequence> to <end sequence>

This is logged whenever a resend request has been sent.

<feed_id> is disconnected. The feed instance has changed.

This is logged if the price publisher has been restarted or if a backup publisher has taken over processing from the primary.

<feed_id> Gaps=%u TotalGaps=%u TrueGaps=%u TrueTotalGaps=%u

Logged at most once per second when data has been dropped.

For example:
<158> Gaps=12 TotalGaps=16 TrueGaps=0 TrueTotalGaps=0
<158> Gaps=13 TotalGaps=18 TrueGaps=0 TrueTotalGaps=0

The gap count moved from 12 to 13, so one instance of dropped data occurred. The total gaps moved from 16 to 18. 18-16 = 2 packets were dropped during the gap. The true gap count did not increment and both dropped packets were recovered

<feed_id> snapshot completed addr=<ipaddress> %u packets decoded

Logged when the snapshot for a given feed has completed.

<feed_id> is disconnected. Snapshot request was rejected.

Logged when snapshot for a given feed is rejected owing to user authentication issues.

<feed_id> is disconnected. Snapshot request has failed.

Logged when snapshot for a given feed has failed.

<feed_id> is now in sequence. Sequence=%u.

Logged when the feed is back in sequence.

<feed_id> is out of sequence.

Logged when the feed has gone out of sequence.

<feed_id> is now in sequence but data has been lost. Old sequence=%u new sequence=%u

Logged when the feed is back in sequence but true gaps in data have occurred.

<feed_id> has not responded in %u m/s. Marking it as down.

Logged when a feed has not sent out a price or a heartbeat in the time interval defined by MdFeedInit.

<feed_id> is suspect, it is not sending discovery heartbeats.

Logged if a feed has not sent out a heartbeat message in uiHeartbeatDC seconds.

<feed_id> is not longer suspect, we received a discovery heartbeat.

Logged when a heartbeat has been received and the feed is no longer in a suspect state.

12. Appendix A

Here is an example of callback sequence you could expect from mdclient API for POSIT Auction.

Upon first connected, snapshots are provided for last known state of symbols:

OnImbalance() for symbol A
OnPriceRecord() for symbol A
OnImbalance() for symbol B
OnPriceRecord() for symbol B
…

NOTE: You may get nothing if there was no activity on any symbol prior to your connection.

As orders come in, whenever a match can be made,

OnImbalance() with imbalance.crossType = 4 (market_imbalance), imbalance.currentReferencePrice = 512(integer representation of 5.12, see Section “Decoding a price”) and imbalance.pairedShares = 100

Indicative size changes to 200,

OnImbalance() with imbalance.crossType = 4 (market_imbalance), imbalance.currentReferencePrice = 512 and imbalance.pairedShares = 200

If orders are canceled mid-auction and a match can no longer be made,

OnImbalance() with imbalance.crossType = 4 (market_imbalance), imbalance.currentReferencePrice = 0 and imbalance.pairedShares = 0

If an auction ends with a match,

OnTrade() with last price and size set, and tradeIndicator equals to trade_indicator_continuous_auction.

If an auction ends without a match,

OnImbalance() with imbalance.crossType = 0 (none), imbalance.currentReferencePrice = 0 and imbalance.pairedShares = 0

- END -