NGM FIX Protocol

Version 1.18.0

6 May 2020

Contents

Chapter 1

Overview

The NGM FIX protocol is the main protocol for communicating with the NGM trading system. The following standard protocols are used:

- FIX 5.0 Service Pack 2 for application level messages.
- FIX session protocol FIXT 1.1 for maintaining FIX sessions.
- FAST 1.1 (FIX Adapted for STreaming) is used for encoding FIX messages, meaning that the traditional ASCII encoding ("Tag=Value") is not supported.
- FAST SCP 1.1 (Session Control Protocol), level 2 (hello, alert and reset messages) is used for managing FAST sessions.
- TCP is used as the underlying reliable transport protocol.

Two services are offered to the user; a private service for order management, order status, trade reporting and similar tasks, and a public service for market data, reference data and other information. Message filtering allows a user to limit which messages can be sent or will be received on a service.

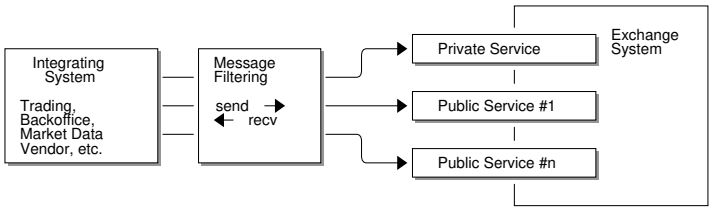

Figure 1.1: Service overview. All connections are FIX/FAST over TCP/IP.

1.1 About this Document

The reader of this document should be somewhat familiar with the FIX protocol. Any non-standard FIX fields or changes from the FIX standard are clearly

highlighted. Whenever we felt that the FIX protocol specification is unclear or something must be bilaterally agreed it is also described in this document. Note that the *type* column in the message tables contains the FAST data type that is used (see chapter 5).

Chapter 1 (this chapter) gives an overview of the NGM FIX protocol.

Chapter 2 describes the parts of the protocol that are common across all services, including the session layer.

Chapter 3 explains the private service which is used for orders, quotes and trades.

Chapter 4 explains the public service which is used for dissemination of market data and reference data.

Chapter 5 describes how FAST is applied to the FIX messages and details about data types.

Chapter 2

General Service Information

This chapter describes the parts of the protocol that are common across all services.

2.1 Recovery

During session initialization, message gaps can occur. These are detected by observing the message sequence number. In these cases two recovery mechanisms are supported; message recovery and full snapshot recovery. Message recovery is the preferred way to quickly recover a few lost messages. In certain cases a session reset is required, e.g. too long time since last connection or disaster recovery (e.g. lost session state). *After a reset the client must do a full snapshot recovery.*

Message recovery is only accepted during logon by observing the *NextExpectedSeqNum* field. Note that the *ResendRequest* message is not supported. See section 2.5.1 for more information and message scenarios.

During full snapshot recovery the client should expect unsolicited updates mixed with snapshot replies, especially if a snapshot is requested intra-day. It is guaranteed that the last message received is always the most recent one, regardless if it is a snapshot reply or an unsolicited update.

2.2 Filtering

For users requiring limited information, functionality or privileges, filtering can be applied to control what can be sent by the exchange or the user. Filtering configuration is performed by contacting the exchange manually.

For each data class, the following filter rules exist:

All The user can send operations, receive live changes and request snapshots. This is the default.

Read-only The user can only receive live changes and request snapshots.

None The user cannot send operations nor receive any data.

Snapshot-only The user can only request snapshots. This is a last resort, since we prefer that users can handle live data or do not receive it at all.

Unauthorized operations will be rejected with the *Business Message Reject* message with *BusinessRejectReason* set to 6 (Not Authorized).

All messages are sent to all users in the trader group with the exception of snapshot replies and session control messages (logon replies and such). As such clients should be aware they will receive the replies (execution reports, trade capture reports and so forth) generated by their peers activities in the market. If this is undesired the user should be in its own trader group or use filtering. Having a private trader group is used if one user does not wish to get information about his peers activities in the market but only his own. Filtering is used if the user wishes to see only certain information, for example only trades, but from all users in the trader group.

What messages are included in each chapter is defined in the messages overview section in each service chapter.

2.2.1 User filtering parameters

User filtering parameters offer a fine grained control of which information a user is allowed to view and manipulate. A typical example is access to a subset of the available market segments.

Unauthorized operations will be rejected in the same way as for the coarse grained filtering, as described above.

2.3 Throughput Limit

2.3.1 Message Throughput

Each user have a throughput limit set, which limits the number of messages that can be sent to the exchange per second. The throughput counter is reset each second (i.e. not a sliding window), and when the throughput exceeds the limit any additional messages are *delayed* until the next second.

The delaying of the operations is performed at the TCP level, resulting in queues first in the exchange TCP buffer, then in the client side TCP buffer and finally in the client side application code. This means that the easiest way of avoiding delays is simply not to exceed the throughput limit. Continuous monitoring of the delay of operations is another approach.

The throughput limit that is used for your user is only available *offline* (outside the protocol), i.e. contact the exchange for more information.

2.3.2 Snapshot Throughput

Apart from the message throughput limit there is an additional snapshot limitation. Each user has a certain number of snapshot requests it can perform per hour. The snapshot counter is reset after one hour, starting from the point in time where the first snapshot was performed.

Note that the snapshot thoughput limitation is applied per snapshot type, thus exceeding the limit of a certain snapshot will not affect other snapshot types. For available snapshot types see section 3.3 and 4.1.

If a user exceeds the limit for a specific snapshot type within the time period, the exchange will reply with a *Business Message Reject*.

2.4 Component Blocks

2.4.1 Standard Header

The *Standard Header* is included in all FIX messages.

For inbound messages (to NGM):

- *SenderCompID* denotes the NGM trader group id or public group id.
- *SenderSubID* is intended for the client side user name.
- *TargetCompID* should be set to "NGM".
- *TargetSubID* should be absent.

For outbound messages (from NGM):

- *SenderCompID* is set to "NGM".
- *SenderSubID* is absent.
- *TargetCompID* denotes the NGM trader group id or public group id.
- *TargetSubID* same as *SenderSubID* in the operation that produced this outbound message.

For inbound messages when sending messages via third party firm(*service connection*).

- *SenderCompID* denotes the NGM trader group id or public group id of service connection.
- *SenderSubID* is intended for the client side user name of service connection.
- *TargetCompID* should be set to "NGM".
- *TargetSubID* should be absent.
- *OnBehalfOfCompID* denotes the NGM trader group id or public group id of the origin firm.
- *OnBehalfOfSubID* is intended for the origin client side user name.

For outbound messages(from NGM) when addressing a member via a third party firm(*service connection*):

- *SenderCompID* is set to "NGM".
- *SenderSubID* is absent.
- *TargetCompID* denotes the NGM trader group id or public group id of service connection.
- *TargetSubID* same as *SenderSubID* in the operation that produced this outbound message.
- *DeliverToCompID* same as *OnBehalfOfCompID* in the operation that produced this outbound message.

- *DerliverToSubID* same as *OnBehalfOfSubID* in the operation that produced this outbound message.

Most messages from the public service are not directed to a specific user and will have *TargetCompID* set to "NGM".

When the *CompID* fields denote a NGM trader group id it has the form *<firm>-<trader group>*, where *<firm>* is the marketplace assigned member code.

When the *CompID* fields denote a NGM public group id it has the form *PUB-<id>*.

The *SubID* fields are intended for client side user traceability and can be set to any value. In the case of unsolicited messages *TargetCompID* and *TargetSubID* are set to the trader group and user that last touched the order etc. that caused the message.

Table 2.1: StandardHeader.

Tag	*Field Name*	*Type*	*Req*	*Description*
34	MsgSeqNum	uInt64	Y	Message sequence number.
49	SenderCompID	string	Y	Identifies sender firm (and trader group).
50	SenderSubID	string	N	Identifies sender user.
56	TargetCompID	string	Y	Identifies target firm (and trader group).
57	TargetSubID	string	N	Identifies target user.
115	OnBehalfOfCompID	string	N	Identifies sending firm, used when sending messages via a third party.
116	OnBehalfOfSubID	string	N	Identifies sending user, used when sending messages via a third party.
128	DeliverToCompID	string	N	Identifies target firm, used when sending messages via a third party.
129	DeliverToSubID	string	N	Identifies target user, used when sending messages via a third party.
52	SendingTime	uInt64	Y	UTC timestamp in microseconds. Time of original message transmission.

2.4.2 Security Ref

The *Security Ref* component block is used to identify a security. Securities (order books) are always identified by a marketplace assigned identifier. This identifier is, together with other identifiers (e.g. ISIN and symbol), published in *Security Definition Update Report* and *Security List* messages.

Table 2.2: SecurityRef.

Tag	*Field Name*	*Type*	*Req*	*Description*
48	SecurityID	string	N	Security identifier of type specified in SecurityIDSource.
22	SecurityIDSource	uInt32	N	Identifies the class of SecurityID. ASCII char enumeration. 'M'=Marketplace-assigned identifier

2.5 Session Messages

The standard FIX transport is used for maintaining FIX sessions with some exceptions.

24/7 connectivity is supported but *MsgSeqNum* is never reset during a connection. This means that *SequenceReset* with reset is not supported, nor is exchange of *Logon* messages during a session (i.e. after the first *Logon*). The *MsgSeqNum* may be reset (to 1) during logon if desired. The *MsgSeqNum* is represented as a 64 bit integer.

The *NextExpectedSeqNum* field is used to resynchronize a FIX session upon logon. Because of this and due to the fact that TCP is used as the underlying (reliable) transport protocol the *ResendRequest* message is not needed nor supported.

Note that if no *Logon* message is received within a certain time, the connection will be closed.

2.5.1 Logon (A)

The *Logon* message is used to initiate a FIX session. When connecting to NGM the following values should be set as follows:

HeartBeatInterval 10 seconds.

SenderCompID As with all messages this should be set to the Tradergroup name (e.g "FOO-1").

Username Specifies the unique user to logon (e.g "FOO-1-1"). The *SenderName* of the FAST *Hello* message must be set to the same value.

The *Logon* message is a part of the message recovery mechanism. The *NextExpectedSeqNum* field is used to resynchronize a FIX session upon logon. By observing this field each party can detect which messages need to be resent to the other party.

If the acceptor (the exchange) detects an error/mismatch in the *Logon* message received it replies with a *Logout* message with any of the following *SessionStatus* values:

Session state is lost, see Section 2.1.

Message recovery not available, i.e. the initiator need messages too far in the past to be resent.

NextExpectedSeqNum is too high, i.e. session state is broken. This indicates some kind of error (e.g. software error, human error).

MsgSeqNum is too low, i.e. session state is broken. This indicates some kind of error (e.g. software error, human error).

Incorrect reset, i.e. sequence number is not set to one when resetting the session.

If the initiator receives any of these errors from the acceptor or detects an error/mismatch in the *Logon* message received it should disconnect and reconnect with logon reset followed by a full snapshot recovery. The last two *SessionStatus* codes indicates some other problem that should also be investigated, but the same recovery procedure is still valid.

Figure 2.1 shows an example logon scenario. Any messages that need to be resent are sent directly after the logon messages has been exchanged. The *Logon* message with *MsgSeqNum=123* is resent as a gap-fill directly after the messages 90-122 have been resent.

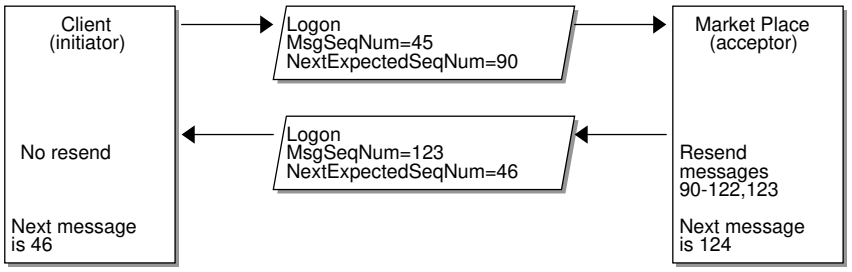

Figure 2.1: Logon procedure with automatic retransmission of messages.

If the initiator want to reset the session it can logon with the *ResetSeqNumFlag* set (see figure 2.2). The *MsgSeqNum* must then also be reset to 1 in the initiator's *Logon* message. The acceptor will also respond with the *ResetSeqNumFlag* set and *MsgSeqNum* set to 1. From that point on both parties will continue with sequence number 2.

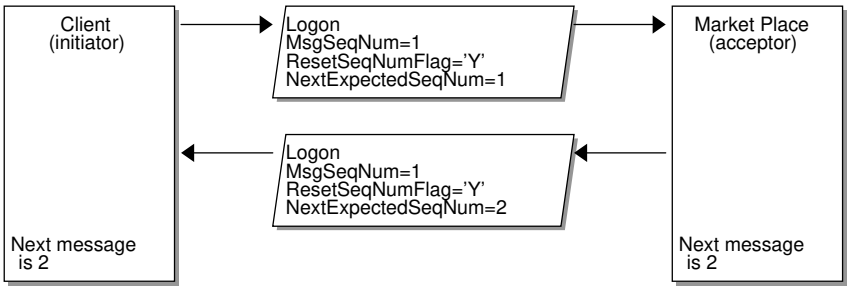

Figure 2.2: A reset requested by the initiator.

Logon:

- is replied to with a *Logon* message
- can be rejected with a *Logout* message, with SessionStatus set to 5 (InvalidUsernameOrPassword), 6 (AccountLocked), 7 (NotAllowed), 100 (HistoryNotAvailable), 9 (ReceivedMsgSeqNumTooLow), 10 (ReceivedNextExpectedMsgSeqNumTooHigh), 103 (SessionStateLost) or 104 (MsgSeqNumNotOne)
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to A
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the Logon message

Logon is sent:

- in reply to a *Logon* message

Table 2.3: Logon (A).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
98	EncryptMethod	uInt32	Y	Method of encryption. 0=None / Other
108	HeartBtInt	uInt32	Y	Heartbeat interval (seconds).
1137	DefaultApplVerID	string	Y	Valid value: "FIX50SP2".

Table 2.3: Logon (A).

Tag	*Field Name*	*Type*	*Req*	*Description*
141	ResetSeqNumFlag	uInt32	N	Indicates both sides of a FIX session should reset sequence numbers. Absence means 'N'. Boolean (ASCII char enumeration). 'N'=Don't reset 'Y'=Reset
789	NextExpectedMsgSeq-Num	uInt64	Y	Message sequence number gap detection.
553	Username	string	N	
554	Password	string	N	

2.5.2 Logout (5)

The *Logout* message initiates or confirms the termination of a FIX session. The logout initiator should wait for the opposite side to respond with a confirming logout message before disconnecting.

Logout:

- is replied to with a *Logout* message, with SessionStatus set to 4 (LogoutComplete)
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to 5
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the Logout message

Logout is sent:

- in reply to a *Logout* message, with SessionStatus set to 4 (LogoutComplete)
- to reject a *Logon* message, with SessionStatus set to 5 (InvalidUsernameOrPassword), 6 (AccountLocked), 7 (NotAllowed), 100 (HistoryNotAvailable), 9 (ReceivedMsgSeqNumTooLow), 10 (ReceivedNextExpectedMsgSeqNumTooHigh), 103 (SessionStateLost) or 104 (MsgSeqNumNotOne)

Table 2.4: Logout (5).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
1409	SessionStatus	uInt32	N	Session status at time of logout. 4= Session logout complete 5= Invalid username or password 6= Account Locked 7= Logons are not allowed at this time 9= Initiators MsgSeqNum is too low. 10= Initiators NextExpectedMsgSeqNum is too high. 100= Requested history is not available. 103= Acceptor has lost the session state. 104= Initiators MsgSeqNum must be equal to one when resetting the session.
58	Text	string	N	

2.5.3 TestRequest (1)

The *Test Request* message is used for requesting a *Heartbeat* message to establish that the session is alive. When receiving a *Test Request*, you should reply with a *Heartbeat* with the TestReqID field set to the value contained in the received *Test Request* message. Note that *Test Request* should not be sent unless it's necessary, that is, when you haven't sent any message (not just *Test Request* and *Heartbeat*) for HeartBtInt seconds.

Any message you send is an indication that you're alive and any message you receive is an indication that the sender is alive.

TestRequest:

- is replied to with a *Heartbeat* message, with TestReqID set to (copied from) the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to 1

- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the TestRequest message

TestRequest is sent:

- unsolicited, when you haven't received any message (not just TestRequest or Heartbeat messages) from the peer for HeartBtInt seconds.

Table 2.5: TestRequest (1).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
112	TestReqID	string	Y	

2.5.4 Heartbeat (0)

Heartbeat sent either unsolicited or as a reply to a *Test Request* message. When receiving a *Heartbeat*, you should not reply to it. This also means that you won't receive a reply from the peer after sending a *Heartbeat*. Note that *Heartbeat* shouldn't be sent unless necessary, that is, when you haven't sent any message (not just *Test Request* and *Heartbeat*) for HeartBtInt seconds.

Any message you send is an indication that you're alive and any message you receive is an indication that the sender is alive.

Heartbeat is sent:

- unsolicited, when you haven't sent any message (not just TestRequest or Heartbeat messages) to the peer for HeartBtInt seconds.
- in reply to a *TestRequest* message, with TestReqID set to (copied from) the value in the request message

Table 2.6: Heartbeat (0).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
112	TestReqID	string	C	Conditionally required when this is a response to a TestRequest.

2.5.5 SequenceReset (4)

The *Sequence Reset* message is only used for sending gap fills during message retransmission.

Table 2.7: SequenceReset (4).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
123	GapFillFlag	uInt32	N	ASCII char enumeration. 'Y'=Gap fill
36	NewSeqNo	uInt64	Y	Next sequence number.

2.5.6 Reject (3)

Session level reject message.

Reject is sent:

- to reject any message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the rejected message

Table 2.8: Reject (3).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
45	RefSeqNum	uInt64	Y	MsgSeqNum of the rejected message.
372	RefMsgType	string	N	The FIX type of the message being referenced.
371	RefTagID	uInt32	N	The FIX field being referenced.
373	SessionRejectReason	uInt32	N	1=Required Tag Missing 5=Value is incorrect (out of range) for this tag 6=Incorrect data format for value 9=CompID problem 10=SendingTime Accuracy Problem 11=Invalid MsgType 14=Tag specified out of required order 99=Other

Table 2.8: Reject (3).

Tag	*Field Name*	*Type*	*Req*	*Description*
58	Text	string	N	Error message.

2.6 General Application Level Messages

2.6.1 Business Message Reject (j)

The *Business Message Reject* message can reject an application-level message which fulfills session level rules and cannot be rejected via any other means.

BusinessMessageReject is sent:

- to reject any message, with BusinessRejectReason set to the reject reason and RefMsgType set to MsgType of the rejected message

Table 2.9: BusinessMessageReject (j).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
372	RefMsgType	string	Y	The MsgType (35) of the FIX message being referenced.
379	BusinessRejectRefID	string	N	The value of the business-level "ID" field on the message being referenced.
380	BusinessRejectReason	uInt32	Y	Code to identify reason for a Business Message Reject message. 0=Other 1=Unknown ID 2=Unknown Security 3=Unknown Message Type 4=Application not available 5=Conditionally required field missing 6=Not Authorized 7=DeliverTo firm not available at this time 18=Invalid price increment

Table 2.9: BusinessMessageReject (j).

Tag	*Field Name*	*Type*	*Req*	*Description*
58	Text	string	N	Where possible, message to explain reason for rejection

Chapter 3

Private Service

The private service is used for sending trading operations to and receiving trading related updates from the exchange. The traffic is of a mixed interactive and non-interactive "multicast" nature. Interactive since information is sent from the exchange in direct response to an operation from the user. Non-interactive since information is also sent spontaneously (not in direct response to a request from the user) from the exchange. Multicast since the same information is sent to a group of users of the service rather than a specific user (drop copies).

Examples of interactive traffic include creation and management of orders and registration of manual trades. Examples of non-interactive traffic include trades (which happen "spontaneously" seen from the perspective of the passive party). An example of multicast traffic includes order updates for orders created by another user in the same trader group. An example of non-multicast traffic is replies to snapshot requests.

As a consequence of the non-interactive and multicast properties of the service, data (typically trades) is pushed to a user's session even when a user is offline. No subscription requests are required nor supported by the service. Instead, a user needs to synchronize with the service when logging on, either on the session level (by requesting retransmission of lost messages) or on the application level (by requesting snapshots).

3.1 User Model

The user model in the private service is divided into three levels (see figure 3.1); organization, trader group and user. Within the organization level orders are matched as internal trades. An organization can have one or more trader groups, which in turn can have one or more users.

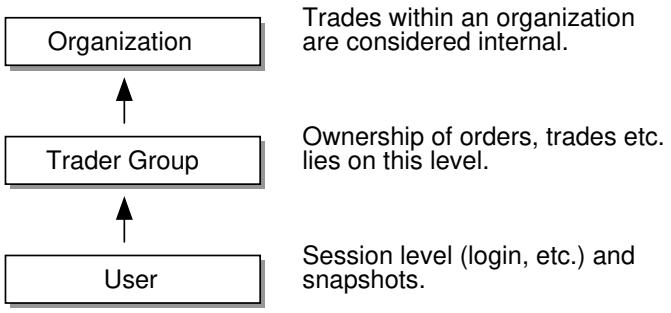

Figure 3.1: User model in the private service.

Ownership of orders and trades lies on the trader group level, and changes to this data is sent to *all* users within the trader group. This means that users within the same trader group can see and modify each other's orders and trades, and receive the result of each others operations.

Each user has a separate FIX session to the private service. A snapshot request will only affect the user that requested it.

For example a backup system (hot standby) should be part of the same trader group as the primary system, and will receive drop copies of the result of the operations that the primary system sends to the exchange.

For example if an organization has two different systems, e.g. one for quotation and another for client orders, they can be put into different trading groups to minimize interference of each other. They will still benefit from internal trades as long as they are part of the same organization.

3.2 Action on Connection Loss

The trading system has a mechanism for handling "unmanaged orders" (and quotes) when a user loses its connection. The mechanism is used to ensure that the organization does not end up in a situation where the market is changing rapidly while the organization has orders or quotes in the market that they are not able to control, because of a network problem, or a hardware crash for example.

The mechanism is activated if a user is disconnected for any reason (except logging out normally) and the disconnected user was the only logged in user in its trader group with order (or quote) managing privileges, which is decided from the filtering settings for the user.

The action performed when the mechanism is activated can be configured individually for each order (see *ExecInst* in the Order component block) and be set to suspend, delete or do nothing with the order. The action for quotes is always delete. The action is only executed if the security is ready to trade (open).

Note that if a client stops sending heartbeat messages as requested it will be disconnected which in turn can trigger the action on connection loss mechanism.

3.3 Full Snapshot Recovery

On the private service snapshots can be requested for the following:

Orders See the *Order Mass Status Request* message in section 3.7.8.

Quotes See the *Quote Status Request* message in section 3.8.6. An alternative is to cancel all quotes instead of requesting a snapshot. However, the time priority of quotes will be lost and all other users within the same trader group will be affected by the quote cancelations.

Trades See the *Trade Capture Report Request* message in section 3.9.6.

3.4 Provider Connection

A FIX connection can serve as a provider connection *'on behalf of'* a member who does not have its' own connection to NGM. One single provider connection may serve multiple members.

The provider connection will use the fields *OnBehalfOfCompID* and *OnBehalfOfSubID* to distinguish the serviced organisations and users when sending messages to the NGM exchange. Outbound messages will contain information in the fields *DeliverToCompID* and *DeliverToSubID* which referes to *OnBehalfOf* fields of the inbound messages.

A provider may only send orders and trades on behalf of another member, thus quotes are not supported.

Note that a provider account needs explicit authorization by NGM for each member and user it will serve as *OnBehalfOf*.

3.4.1 Supported messages

Inbound messages allowed to use *OnBehalfOfCompID* and *OnBehalfOfSubID*:

- NewOrderSingle
- OrderCancelReplaceRequest
- OrderCancelRequest
- TradeCaptureReport

Outbound messages using *DeliverToCompID* and *DeliverToSubID* fields:

- ExecutionReport
- TradeCaptureReport
- OrderCancelReject
- BusinessMessageReject

3.5 Message Overview

The following messages can be sent/received by the client to/from the private service. Depending on the filter rules only a subset of the following messages may be sent/received.

Table 3.1: Message overview.

Message	*Class*	*All?*	*Read-only?*	*Snap-shot-only?*
NewOrderSingle	Order	send		
OrderCancelReplaceRequest	Order	send		
OrderCancelRequest	Order	send		
ExecutionReport	Order	recv	recv	recv
OrderCancelReject	Order	recv	recv	
OrderMassStatusRequest	Order	send	send	send
Quote	Quote	send		
QuoteCancel	Quote	send		
QuoteStatusReport	Quote	recv	recv	recv
QuoteRequest	Quote	recv	recv	
QuoteStatusRequest	Quote	send	send	send
TradeCaptureReport	Trade	send recv	recv	recv
TradeCaptureReportAck	Trade	recv	recv	
TradeCaptureReportRequest	Trade	send	send	send
TradeCaptureReportRequestAck	Trade	recv	recv	recv
UserSecurityStatusUpdateRequest	Security status			
UserSecurityStatusUpdateResponse	Security status			

3.5.1 Filtering Examples

The following are examples of filter rules that could suit certain systems that do not wish to receive all data.

Back-office system that only need drop copies of trades from other users in the same trader group:
Order=none, Quote=none, Trade=read-only.

Mass quoting system that do not need to see (client) orders nor submit manual trades:
Order=none, Quote=all, Trade=read-only.

Client order system that only manage client orders (not quotes) and that do submit manual trades:
Order=all, Quote=none, Trade=all.

3.6 Parties Information

Orders, quotes and trades contains parties information. However, not all combinations of parties are used in all situations, see table below. Party roles used for order and quotes are also copied into Trade Capture Reports when orders or quotes are matched.

Also see chapter A.2.

Table 3.2: Usage of parties information.

PartyRole	*Usage*	*PartyID-Source*	*Party-Role-Qual?*	*Party-SubId?*
ClientID (3)	Order	Short (P)	Y	N
Executing trader (12)	Order/Quote	Short (P)	Y	N
Investment decision maker (122)	Order/Quote	Short (P)	Y	N
Entering Firm (7)	From exchange	Custom (D)	N	Y
Contra Firm (17)	Trade	Custom (D)	N	Y
Buyer/Seller (27)	Trade	Custom (D)	N	Y

3.6.1 Parties Component Block

This component block is used to specify parties.

Table 3.3: Parties.

Tag	*Field Name*	*Type*	*Req*	*Description*
453	PartyIDs	sequence	N	
452	→PartyRole	uInt32	Y	3=ClientID 12=Executing trader 122=Investment decision maker 7=Entering Firm (identifies the provider in an onBehalfOf scenario) 17=Contra Firm 27=Buyer/Seller
2376	→PartyRoleQualifier	uInt32	N	22=Algorithm 23=Firm or legalEntity 24=Natural person

Table 3.3: Parties.

Tag	*Field Name*	*Type*	*Req*	*Description*
447	→PartyIDSource	uInt32	Y	ASCII char enumeration. 'D'=Proprietary/custom code (marketplace assigned member id) 'P'=Short code identifier, represented as an unsigned 64-bit integer. Short code translation must be reported outside protocol
448	→PartyID	string	Y	
802	→PartySubIDs	sequence	N	
803	→→PartySubIDType	uInt32	Y	Used to indicate the counter party trader ID in TradeCaptureReport when TradeHandlingInstr='3'. Also used to further identify entering firm. 2=Person 3=System (trader group)
523	→→PartySubID	string	Y	

3.7 Order Messages

An order can be identified in a number of ways:

ClOrdID Client assigned identifier (mandatory). It must be unique within a security and trader group. This identifier must change each time the client updates the order and thus denotes a revision of the order.

OrderID Market place assigned identifier which does not change during the lifetime of the order.

SecondaryOrderID Reference to the current *MDEntryID* in the market data which identifies the order. This identifier is only present for orders that are visible in the market data and it may change whenever the order is seen as a new order in the market data (e.g. refills of iceberg orders).

Either *OrigClOrdID* or *OrderID* is required for order modification and deletion. Usage of *OrigClOrdID* allows for chaining of order operations.

3.7.1 Order Component Block

This component block is used to define an order.

Table 3.4: Order.

Tag	*Field Name*	*Type*	*Req*	*Description*
54	Side	uInt32	Y	ASCII char enumeration. '1'=buy '2'=sell
40	OrdType	uInt32	C	ASCII char enumeration. '1'=market '2'=limit Required in NewOrderSingle. Required in OrderCancel-ReplaceRequest.
44	Price	decimal	N	Required for limit orders.
38	OrderQty	decimal	C	Required in NewOrderSingle. Required in OrderCancel-ReplaceRequest.
1138	DisplayQty	decimal	N	Displayed quantity on iceberg/reserve order.
1083	DisplayWhen	uInt32	N	Instructs when to refresh DisplayQty. ASCII char enumeration. '1'=Immediate (after each fill) '2'=Exhaust (when DisplayQty = 0)
1084	DisplayMethod	uInt32	N	Defines what value to use in DisplayQty. If not specified the default DisplayMethod is '1'. ASCII char enumeration. '1'=Initial (use original DisplayQty) '2'=New (use RefreshQty) '3'=Random (randomize value)
1088	RefreshQty	decimal	C	Conditionally required when DisplayMethod is '2' (New).
1085	DisplayLowQty	decimal	C	Conditionally required when DisplayMethod is '3' (Random).
1086	DisplayHighQty	decimal	C	Conditionally required when DisplayMethod is '3' (Random).

Table 3.4: Order.

Tag	*Field Name*	*Type*	*Req*	*Description*
1087	DisplayMinIncr	decimal	N	Can be used to specify larger increments than the standard increment provided by the market (round lot size) when DisplayMethod='3'.
59	TimeInForce	uInt32	N	Absence means '0'. ASCII char enumeration. '0'=Session '1'=Good Till Cancel(GTC) '3'=Immediate Or Cancel (IOC) '4'=Fill Or Kill (FOK) '6'=Good Till Date (GTD)
126	ExpireTime	uInt64	C	UTC timestamp. Conditionally required when TimeInForce is '6' (Good Till Date).
60	TransactTime	uInt64	N	UTC timestamp this order request was created, updated or cancelled.
1	Account	string	N	Account information that will be echoed back.
18	ExecInst	string	N	Instructions for order handling (separated with spaces). Valid values: S=Suspend o=Cancel on connection loss (mutually exclusive with p) p=Suspend on connection loss (mutually exclusive with o)
529	OrderRestrictions	string	N	Multiple char value (delimited with space). Restrictions associated with an order. B=Issuer Holding C=Issue Price Stabilization

3.7.2 Order Attributes Grp Component Block

This component block defines additional order attributes.

Table 3.5: OrderAttributeGrp.

Tag	*Field Name*	*Type*	*Req*	*Description*
2593	OrderAttributes	sequence	N	
2594	→OrderAttributeType	uInt32	Y	2=Liquidity provision activity order (when together with OrderAttributeValue=Y, it signifies that the order was submitted "as part of market making strategy pursuant to articles 17 and 18 of Directive 2014/65/EU"). 3=Risk reduction order (when together with OrderAttributeValue=Y, it signifies that the commodity derivative order is a transation "to reduce risk in an objectively measurable way in accordance with Article 57 of Directive 2014/65/EU"). 5=Systematic internalizer order (when together with OrderAttributeValue=Y, it signifies that the order is submitted by a systematic internalizer).
2595	→OrderAttribute-Value	string	Y	The value associated with the attribute type specified in OrderAttributeType. Must be "Y".

3.7.3 New Order Single (D)

The *New Order Single* message is used to create a new order. The response is always an *Execution Report*, including rejects.

NewOrderSingle:

- is replied to with an *ExecutionReport* message, with ClOrdID set to (copied from) the value in the request message
- can be rejected with an *ExecutionReport* message, with ExecType set to '8' (Rejected) and ClOrdID set to (copied from) the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to D
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the NewOrderSingle message

Table 3.6: NewOrderSingle (D).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
11	ClOrdID	string	Y	
	component block <SecurityRef>			Security identification.
	component block <Order>			
528	OrderCapacity	uInt32	N	Designates the capacity of the firm placing the order. Absence means 'R'. 'P'=Principal (Deal) 'R'=Riskless principal (Matched) 'A'=Agency (Any other capacity)
1724	OrderOrigination	uInt32	N	Identifies the origin of the order. Absence means non DEA. '5'=Order received from a direct access or sponsored access customer
	component block <OrderAttributeGrp>			
	component block <Parties>			

3.7.4 Order Cancel/Replace Request (G)

The *Order Cancel/Replace Request* (a.k.a. *Order Modification Request*) is used to replace an *existing* order (i.e. not filled or removed). Side or security cannot be changed in an order.

The modification is replied to with an *Execution Report* if successful. Otherwise an *Order Cancel Reject* message is sent.

OrderCancelReplaceRequest:

- is replied to with an *ExecutionReport* message, with ClOrdID set to (copied from) the value in the request message
- can be rejected with an *OrderCancelReject* message, with ClOrdID set to (copied from) the value in the request message and CxlRejReason set to the reject reason
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to G
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the OrderCancelReplaceRequest message

Table 3.7: OrderCancelReplaceRequest (G).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
37	OrderID	string	C	Conditionally required when OrigClOrdID is absent.
41	OrigClOrdID	string	C	Conditionally required when OrderID is absent.
11	ClOrdID	string	Y	
	component block <SecurityRef>			Security identification. Must match original order.
	component block <Order>			Side must match original order.

3.7.5 Order Cancel Request (F)

The *Order Cancel Request* is used to cancel an existing order.

The cancelation is replied to with an *Execution Report* if successful. Otherwise an *Order Cancel Reject* message is sent.

OrderCancelRequest:

- is replied to with an *ExecutionReport* message, with ClOrdID set to (copied from) the value in the request message
- can be rejected with an *OrderCancelReject* message, with ClOrdID set to (copied from) the value in the request message and CxlRejReason set to the reject reason
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to F
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the OrderCancelRequest message

Table 3.8: OrderCancelRequest (F).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
37	OrderID	string	C	Conditionally required when OrigClOrdID is absent.
41	OrigClOrdID	string	C	Conditionally required when OrderID is absent.
11	ClOrdID	string	Y	
	component block <SecurityRef>			Security identification. Must match original order.
60	TransactTime	uInt64	Y	UTC timestamp this order was cancelled.

3.7.6 Execution Report (8)

If an order is (partially) filled upon hitting the order book only one *Execution Report* will be sent, with execution type *New* and order status *(Partially) Filled*. For partially filled IOC (Immediate or cancel) and FoK (Fill or kill) orders that are executed directly, one *Execution Report* will be generated with execution type *New* and order status *Cancelled* where the field *CumQty* holds the partial fill volume.

When *WorkingIndicator* is set to 'N', the order operation has been received but not yet executed. In this case any (partially) fills are delayed until the *WorkingIndicator* is changed to 'Y'. An order with *WorkingIndicator* set to 'N' can be modified and deleted as normal.

In case of multiple fills of an order in a single match operation, only one *Execution Report* will be sent for all partial fills. Pending order states are not used. Also the *Done for day* state is never sent for orders, since this can be concluded by observing the security status.

In case of a canceled trade, any orders that were part of the trade will not be restated. The trade cancel is notified only through a *Trade Capture Report* message, no *Execution Report* message is sent.

ExecutionReport is sent:

- unsolicited, when the order is updated, for example when it is part of a matching operation or expires
- in reply to a *NewOrderSingle* message, with ClOrdID set to (copied from) the value in the request message
- to reject a *NewOrderSingle* message, with ExecType set to '8' (Rejected) and ClOrdID set to (copied from) the value in the request message
- in reply to an *OrderCancelReplaceRequest* message, with ClOrdID set to (copied from) the value in the request message
- in reply to an *OrderCancelRequest* message, with ClOrdID set to (copied from) the value in the request message
- in reply to an *OrderMassStatusRequest* message, with MassStatusReqID set to (copied from) the value in the request message and ExecType set to 'I' (OrderStatus)

Table 3.9: ExecutionReport (8).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
17	ExecID	string	Y	Unique identifier of execution message, or "0" for ExecType='I' (Order Status).
150	ExecType	uInt32	Y	ASCII char enumeration. '0'=New '4'=Canceled '5'=Replaced '8'=Rejected '9'=Suspended 'C'=Expired 'F'=Trade (partial fill or fill) 'I'=Order Status
	component block <SecurityRef>			Security identification.
	component block <Order>			
37	OrderID	string	Y	
278	MDEntryID	string	N	Reference to the MDEntryID of this order in the market data.

Table 3.9: ExecutionReport (8).

Tag	*Field Name*	*Type*	*Req*	*Description*
11	ClOrdID	string	N	Conditionally required when this message is a response to a submitted order.
41	OrigClOrdID	string	N	Conditionally required when not unsolicited and ExecType is '4' (Canceled) or '5' (Replaced).
39	OrdStatus	uInt32	Y	ASCII char enumeration. '0'=New '1'=Partially filled '2'=Filled '4'=Canceled '8'=Rejected '9'=Suspended 'C'=Expired
636	WorkingIndicator	uInt32	N	Indicates if the order is currently being worked. Applicable for OrdStatus = "New" and OrdStatus = "Partially filled". Absense means 'Y'. ASCII char enumeration (boolean). 'Y'=Order is currently being worked. 'N'=Order has been accepted but not yet in a working state.
151	LeavesQty	decimal	Y	
14	CumQty	decimal	Y	
1093	LotType	uInt32	N	Defines the lot type assigned to the order. ASCII char enumeration. '1'=Odd Lot '2'=Round Lot
6	AvgPx	decimal	N	Average traded price.

Table 3.9: ExecutionReport (8).

Tag	*Field Name*	*Type*	*Req*	*Description*
103	OrdRejReason	uInt32	N	Code to identify reason for order rejection. 1=Unknown symbol 2=Exchange closed 5=Unknown order 6=Duplicate Order (e.g. dupe ClOrdID) 18=Invalid price increment 99=Other 100=Orders not allowed in knockout or knockout soft state 101=Buy orders not allowed in knockout buyback state 102=Suspended orders not allowed in knockout buyback state 103=Buy orders not allowed in buyback state 104=Sell orders not allowed in distribution state 105=Suspended orders not allowed in distribution state 106=Order not allowed to breach circuit breaker 107=Order breached pre trade control price limit 108=Order breached pre trade control value limit 109=Value less than reserve order minimum value. 110=Reserve order not allowed. 111=Order breached pre trade control volume limit 112=Order for this specific instrument and/or member is blocked by a killswitch

Table 3.9: ExecutionReport (8).

Tag	*Field Name*	*Type*	*Req*	*Description*
378	ExecRestatement-Reason	uInt32	N	Code to identify reason for an Execution Report message sent when communicating an unsolicited cancel. 0=GT corporate action 99=Other
20028	OrderPriority	uInt64	N	Indicates the priority of the order in the order-book in comparison to other orders on the same level. Higher value means lower priority. Custom field.
528	OrderCapacity	uInt32	N	Designates the capacity of the firm placing the order. 'P'=Principal (Deal) 'R'=Riskless principal (Matched) 'A'=Agency (Any other capacity)
1724	OrderOrigination	uInt32	N	Identifies the origin of the order. Absence means non DEA. '5'=Order received from a direct access or sponsored access customer
	component block <OrderAttributeGrp>			
	component block <Parties>			
584	MassStatusReqID	string	N	Value assigned by issuer of Mass Status Request to uniquely identify the request.
912	LastRptRequested	uInt32	N	Indicates that this is the last Execution Report which will be returned as a result of the request. ASCII char enumeration (boolean). 'N'=Not Last Message 'Y'=Last Message
58	Text	string	N	Error message.

3.7.7 Order Cancel Reject (9)

This message is sent in response to *Order Cancel (Replace) Request* in case of an error.

OrderCancelReject is sent:

- to reject an *OrderCancelRequest* message, with ClOrdID set to (copied from) the value in the request message and CxlRejReason set to the reject reason
- to reject an *OrderCancelReplaceRequest* message, with ClOrdID set to (copied from) the value in the request message and CxlRejReason set to the reject reason

Table 3.10: OrderCancelReject (9).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
37	OrderID	string	Y	If CxlRejReason=*Unknown Order*, value is "[N/A]".
41	OrigClOrdID	string	Y	ClOrdId of the order that could not be canceled/replaced.
11	ClOrdID	string	Y	Same as in the request.
39	OrdStatus	uInt32	Y	If CxlRejReason=*Unknown Order*, value is '8'. ASCII char enumeration. '0'=New '1'=Partially filled '2'=Filled '3'=Done for day '4'=Canceled '8'=Rejected '9'=Suspended 'C'=Expired
434	CxlRejResponseTo	uInt32	Y	Identifies type of message this reject is in response to. ASCII char enumeration. '1'=Order cancel request '2'=Order cancel/replace request

Table 3.10: OrderCancelReject (9).

Tag	*Field Name*	*Type*	*Req*	*Description*
102	CxlRejReason	uInt32	N	1=Unknown order 6=Duplicate ClOrdID (11) received 18=Invalid price increment 99=Other 100=Orders not allowed in knockout or knockout soft state 101=Buy orders not allowed in knockout buyback state 102=Suspended orders not allowed in knockout buyback state 103=Buy orders not allowed in buyback state 104=Sell orders not allowed in distribution state 105=Suspended orders not allowed in distribution state 107=Order breached pre trade control price limit 108=Order breached pre trade control value limit 109=Value less than reserve order minimum value. 110=Reserve order not allowed. 111=Order breached pre trade control volume limit 112=Order for this specific instrument and/or member is blocked by a killswitch
58	Text	string	N	Error message.

3.7.8 Order Mass Status Request (AF)

Status for all orders owned by the requester's trader group can be requested with the *Order Mass Status Request* message where *MassStatusReqType* is set to 7 (Status for all orders). This message will be replied to with one or more *Execution Report* messages with *ExecType* set to 'I' (Order Status). The last *Execution Report* will always be indicated with *LastRptRequested* field set to

'Y'. Note that a dummy *Execution Report* with *OrderID* set to "[N/A]" and *LastRptRequested* field set to 'Y' may be sent as last message to indicate the request has been processed (for example as a reply with no orders).

In the event of a malformed request, the response will be a *Business Message Reject* message.

OrderMassStatusRequest:

- is replied to with an *ExecutionReport* message, with MassStatusReqID set to (copied from) the value in the request message and ExecType set to 'I' (OrderStatus)
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to AF
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the OrderMassStatusRequest message

Table 3.11: OrderMassStatusRequest (AF).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
584	MassStatusReqID	string	Y	
585	MassStatusReqType	uInt32	Y	7=Status for all orders

3.8 Quote Messages

A quote can be identified in a number of ways:

QuoteMsgID Client assigned identifier (mandatory). It must be unique within a security and trader group. This identifier must change each time the client updates the quote and thus denotes a revision of the quote.

QuoteID Market place assigned identifier which does not change during the lifetime of the quote.

BidMDEntryID and OfferMDEntryID Reference to the current *MDEntryID* in the market data which identifies the bid/offer. This identifier is only present for quotes that are visible in the market data and it may change whenever the quote is seen as a new bid/offer in the market data (e.g. price changes).

Either *OrigQuoteMsgID* or *QuoteID* is required for quote modification and deletion. Usage of *OrigQuoteMsgID* allows for chaining of quote operations.

All quotes are tradeable, meaning that they are matched against other orders and quotes in the order book.

Zero spread (same bid and offer prices) quotes are supported and will not result in a trade between the sides of the same quote. Crossing prices are however not supported.

Single side quotes are supported by leaving the opposite price field absent (null), e.g. if *BidPx* is present while *OfferPx* then the quote only have a buy side.

The *Quote* and *Quote Status Report* messages have been extended with *TotalBidSize* and *TotalOfferSize*. The *TotalBidSize* is the total (original) bid volume while *BidSize* is the available bid volume. This means that *TotalBidSize* = *BidSize* + cumulative traded bid volume (including any canceled trades). The volume in tradeable quotes are updated using *TotalBidSize* and *TotalOfferSize*.

In case of a (partial) fill of a quote a *Quote Status Report* is sent with an updated available volume. No *ExecutionReport* is sent for a quote fill. However, a *Trade Capture Report* is always sent for any trades that occur. A completely filled quote is deleted.

All quotes are automatically deleted when the trading session ends (*SecurityTradingStatus* is post open).

During financial status substate *Buyback* the exchange accepts double sided quotes from the market maker, however the sell side of the quote is ignored. This is reflected in the *Quote Status Report* where price (*OfferPx*) and volume (*TotalOfferSize*) of the sell side will be cleared.

3.8.1 Quote Grp Component Block

This component block defines a quote.

Table 3.12: QuoteGrp.

Tag	*Field Name*	*Type*	*Req*	*Description*
132	BidPx	decimal	C	Bid price. Either BidPx, OfferPx or both must be specified. Conditionally required in Quote when OfferPx is absent. Conditionally required in QuoteStatusReport when OfferPx is absent, QuoteStatus is not 4 (Canceled All) or 5 (Rejected) and SecurityID is not absent.

Table 3.12: QuoteGrp.

Tag	*Field Name*	*Type*	*Req*	*Description*
133	OfferPx	decimal	C	Offer price. Either BidPx, OfferPx or both must be specified. Conditionally required in Quote when BidPx is absent. Conditionally required in QuoteStatusReport when BidPx is absent, QuoteStatus is not 4 (Canceled All) or 5 (Rejected) and SecurityID is not absent.
1749	TotalBidSize	decimal	N	Specifies the total bid size.
1750	TotalOfferSize	decimal	N	Specifies the total ask size.
60	TransactTime	uInt64	N	UTC timestamp this quote was created, updated or cancelled.
1	Account	string	N	Account information that will be echoed back.
537	QuoteType	uInt32	N	Identifies the type of quote. Absense means restriced tradeable. Valid values: 1=Tradeable. 4=Initially tradeable (quote validation).
529	OrderRestrictions	string	N	Multiple char value (delimited with space). Restrictions associated with an order. B=Issuer Holding C=Issue Price Stabilization
	component block <Parties>			

3.8.2 Quote (S)

The *Quote* message is used for sending new quotes, updating previous quotes and replying to quote requests.

The response, depending on the *QuoteResponseLevel*, is:

No ack (0) No response.

Negative ack only (1) A *Quote Status Report* with *QuoteStatus* = 5 (rejected) in case of a reject.

Ack each message (2) One *Quote Status Report* with *QuoteStatus* = 0 (accepted), 21 (traded), 22 (traded and removed) or 5 (rejected).

Summary ack (3) Not applicable.

Only *Quote Status Report* messages with *QuoteStatus* = 0 (accepted) is considered an acknowledgement, meaning quotes that immediately becomes (partially) filled will always generate a *Quote Status Report*. When a new quote with the quote validation mechanism enabled is entered (*QuoteType* = 4) or an existing quote is modified to *QuoteType* = 4, the value of *QuoteResponseLevel* is ignored and is always considered to be 2. This exception is to make sure that the client receives the *QuoteID* that is needed to reply to *QuoteRequest* messages.

Quote:

- is replied to with a *QuoteStatusReport* message, with QuoteMsgID set to (copied from) the value in the request message
- can be rejected with a *QuoteStatusReport* message, with QuoteMsgID set to (copied from) the value in the request message and QuoteStatus set to 5 (Rejected)
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to S
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the Quote message

Table 3.13: Quote (S).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
	component block <SecurityRef>			
131	QuoteReqID	string	C	Conditionally required when quote is in response to a QuoteRequest message.
117	QuoteID	string	C	Quote identifier assigned by the exchange. Conditionally required when modifying a quote and OrigQuoteMsgID is absent.

Table 3.13: Quote (S).

Tag	*Field Name*	*Type*	*Req*	*Description*
1166	QuoteMsgID	string	Y	Unique client-assigned identifier for the (replacement) quote.
20018	OrigQuoteMsgID	string	C	Reference to previous QuoteMsgID in case of modification. Custom field. Conditionally required when modifying a quote and QuoteID is absent.
301	QuoteResponseLevel	uInt32	N	Level of response requested. Absence means 2. 0=No Acknowledgement 1=Acknowledge only negative or erroneous quotes 2=Acknowledge each quote message
	component block <QuoteGrp>			

3.8.3 Quote Status Report (AI)

The *Quote Status Report* message is used for replying to quote operations and for sending unsolicited updates of the available volume in case a quote is (partially) filled. Quote status reports in response to an operation can be controlled by setting the *QuoteResponseLevel* field, while unsolicited reports are always sent.

The following list explains what values the *QuoteStatus* field will have when a quote is deleted from the system:

User cancel Canceled (17)

Timeout on Quote Request Expired (7)

Security enters Post Open Expired (7)

Admin delete Removed From Market (6)

Security enters Trade Halt Removed From Market (6)

Security enters any knock out substate Removed From Market (6)

Security leaves knock out buyback Removed From Market (6)

Action on Connection Loss Removed From Market (6)

All volume of the Quote is matched Traded and removed (22)

QuoteStatusReport is sent:

- unsolicited, when the quote is updated, for example when it is part of a matching operation or expires
- in reply to a *Quote* message, with QuoteMsgID set to (copied from) the value in the request message
- to reject a *Quote* message, with QuoteMsgID set to (copied from) the value in the request message and QuoteStatus set to 5 (Rejected)
- in reply to a *QuoteCancel* message, with QuoteStatus set to 4 (CanceledAll) or 17 (Canceled) and QuoteMsgID set to (copied from) the value in the request message
- to reject a *QuoteCancel* message, with QuoteStatus set to 5 (Rejected) and QuoteMsgID set to (copied from) the value in the request message
- in reply to a *QuoteStatusRequest* message, with QuoteStatus set to 8 (Query) and QuoteStatusReqID set to (copied from) the value in the request message

Table 3.14: QuoteStatusReport (AI).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
	component block <SecurityRef>			
117	QuoteID	string	N	Quote identifier.
1166	QuoteMsgID	string	N	Maps to QuoteMsgID of a single Quote.
20018	OrigQuoteMsgID	string	N	Maps to OrigQuoteMsgID of a single Quote. Custom field.
649	QuoteStatusReqID	string	N	
297	QuoteStatus	uInt32	Y	The status of the Quote Status Report. 0=Accepted 4=Canceled All 5=Rejected 6=Removed From Market 7=Expired 8=Query 17=Canceled 21=Traded 22=Traded and removed (both sides)

Table 3.14: QuoteStatusReport (AI).

Tag	*Field Name*	*Type*	*Req*	*Description*
300	QuoteRejectReason	uInt32	N	Reason quote was rejected. 1=Unknown Symbol (security) 2=Exchange (Security) closed 5=Unknown Quote 6=Duplicate Quote 7=Invalid bid/ask spread 8=Invalid price 11=Quote Locked - Unable to Update/Cancel (Missing QuoteReqID) 99=Other 100=Not authorized to quote security with Quote Validation 101=Duplicate quote with Quote Validation 102=Quotes not allowed in knockout or knockout soft state 103=Not authorized to quote security in knockout buyback state 104=Sell quotes not allowed in knockout buyback state 105=Not authorized to quote security in distribution state 106=Buy quotes not allowed in distribution state 107=Not authorized to quote security in buyback state 108=Sell quotes not allowed in buyback state 109=Quote breached pre trade control price limit 110=Quote breached pre trade control value limit 111=Quote breached pre trade control volume limit 112=Quote for this specific instrument and/or member is blocked by a killswitch

Table 3.14: QuoteStatusReport (AI).

Tag	*Field Name*	*Type*	*Req*	*Description*
636	WorkingIndicator	uInt32	N	Indicates if the quote is currently being worked. Applicable when QuoteType is not 4. Absence means 'Y'. ASCII char enumeration (boolean). Field added. 'Y'=Order is currently being worked. 'N'=Order has been accepted but not yet in a working state.
1745	BidMDEntryID	string	N	The MDEntryID of the bid side in the market data.
1746	OfferMDEntryID	string	N	The MDEntryID of the offer side in the market data.
134	BidSize	decimal	N	Specifies the available bid size.
135	OfferSize	decimal	N	Specifies the available ask size.
20029	BidPriority	uInt64	N	Indicates the priority of the bid in the orderbook in comparison to other orders and quotes on the same level. Higher value means lower priority. Custom field.
20030	OfferPriority	uInt64	N	Indicates the priority of the offer in the orderbook in comparison to other orders and quotes on the same level. Higher value means lower priority. Custom field.
	component block <QuoteGrp>			
912	LastRptRequested	uInt32	N	Indicates that this is the last report which will be returned as a result of the request. ASCII char enumeration (boolean). Field added. 'N'=Not Last Message 'Y'=Last Message
58	Text	string	N	Error message.

3.8.4 Quote Cancel (Z)

The *Quote Cancel* message is used for canceling a single quote, all quotes for a single security or all quotes.

The response, depending on the *QuoteResponseLevel*, is:

No ack (0) No response.

Negative ack only (1) A *Quote Status Report* with *QuoteStatus* = 5 (rejected) in case of a reject.

Ack each message (2) If canceling a single quote, one *Quote Status Report* is sent with *QuoteStatus* = 17 (canceled) or 5 (rejected). If canceling multiple quotes, then one *Quote Status Report* with *QuoteStatus* = 4 (canceled all) or 5 (rejected) is sent for the operation itself after the reports for each canceled quote where *QuoteStatus* = 17 (canceled).

Summary ack (3) Only applicable for cancel multiple quotes. One *Quote Status Report* with *QuoteStatus* = 4 (canceled all) or 5 (rejected) is sent.

QuoteCancel:

- is replied to with a *QuoteStatusReport* message, with QuoteStatus set to 4 (CanceledAll) or 17 (Canceled) and QuoteMsgID set to (copied from) the value in the request message
- can be rejected with a *QuoteStatusReport* message, with QuoteStatus set to 5 (Rejected) and QuoteMsgID set to (copied from) the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to Z
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the QuoteCancel message

Table 3.15: QuoteCancel (Z).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
	component block <SecurityRef>			Component added. Conditionally required when QuoteCancelType is 1 (Cancel for a security) or 5 (Cancel quote specified in QuoteID or OrigQuoteMsgID).

Table 3.15: QuoteCancel (Z).

Tag	*Field Name*	*Type*	*Req*	*Description*
131	QuoteReqID	string	C	Conditionally required when quote is in response to a QuoteRequest message.
117	QuoteID	string	C	Quote identifier assigned by the exchange. Conditionally required when QuoteCancelType is 5 (Cancel quote specified in QuoteID or OrigQuoteMsgID) and OrigQuoteMsgID is absent.
1166	QuoteMsgID	string	Y	Unique client-assigned identifier for the request.
20018	OrigQuoteMsgID	string	C	Reference to previous QuoteMsgID. Custom field. Conditionally required when QuoteCancelType is 5 (Cancel quote specified in QuoteID or OrigQuoteMsgID) and QuoteID is absent.
298	QuoteCancelType	uInt32	Y	Identifies the type of quote cancel. 1=Cancel for a security 4=Cancel all quotes 5=Cancel quote specified in QuoteID or OrigQuoteMsgID
301	QuoteResponseLevel	uInt32	N	Level of response requested. Absence means 2. 0=No Acknowledgement 1=Acknowledge only negative or erroneous quotes 2=Acknowledge each quote message 3=Summary Acknowledgement
60	TransactTime	uInt64	N	UTC timestamp this quote was cancelled.

3.8.5 Quote Request (R)

The *Quote Request* message is used by the market place to request an updated quote, when the quote validation mechanism is enabled. The request identifies a single quote that need to be updated. The market maker should respond with a *Quote* message, with updated values or confirming previous values, or with a *Quote Cancel* message. If the market maker does not respond within a pre-defined timeout the quote will be canceled.

QuoteRequest is sent:

- unsolicited, when the quote would be part of a matching operation a update (or cancellation) of the quote is required

Table 3.16: QuoteRequest (R).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
	component block <SecurityRef>			
131	QuoteReqID	string	Y	Unique identifier for quote request.
117	QuoteID	string	Y	Quote identifier. Field added.

3.8.6 Quote Status Request (a)

A snapshot of all quotes can be requested using the *Quote Status Request* message. The response is one or more *Quote Status Report* messages with *QuoteStatus* = 8 (query). The last response has the *LastRptRequested* field set to 'Y'. Note that if there are no quotes available, a dummy quote with no *SecurityID* set (null) will be sent as the last and only message.

QuoteStatusRequest:

- is replied to with a *QuoteStatusReport* message, with QuoteStatus set to 8 (Query) and QuoteStatusReqID set to (copied from) the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to a
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the QuoteStatusRequest message

Table 3.17: QuoteStatusRequest (a).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
649	QuoteStatusReqID	string	N	
263	SubscriptionRequest-Type	uInt32	Y	ASCII char enumeration. '0'=Snapshot

3.9 Trade Messages

Both automatic matching of orders/quotes and manual trades are conveyed using the *Trade Capture Report* message.

For manual trade reporting, one-party report for pass-through to counterparty (figure 3.2), is the only accepted trading model for *non-internal* trades. For internal trades, where the counterparty is the same as the reporting party, the two-party report trading model (figure 3.3) is also accepted. Providers may also use the two-party report trading model, for trades between trader groups for which they are allowed to act on behalf of.

The *Trade Capture Report* message is also used for matching orders in a manually matched orderbook. See appendix B for more information.

In the one-party for pass-through model the initiator can cancel the trade as long as it is not confirmed by the counterparty. Non-confirmed trades have no *TradeID*, which means that they must be referenced to with the *TradeReportRefID* field.

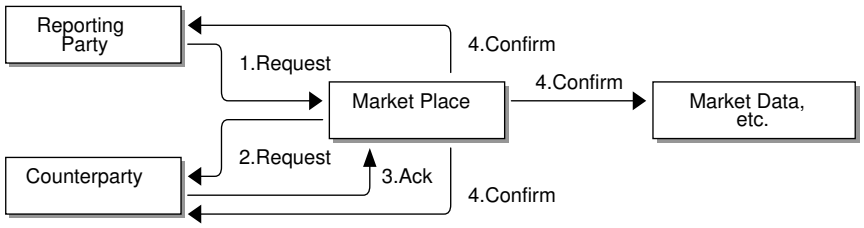

Figure 3.2: Privately negotiated trade, one-party report for pass-through to counterparty.

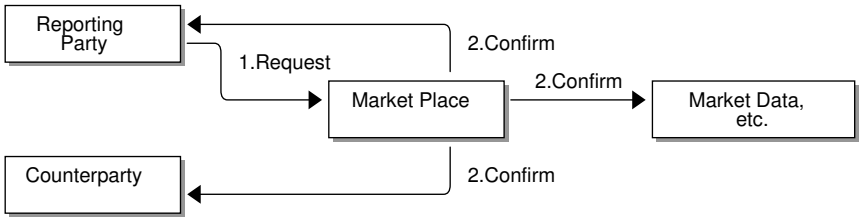

Figure 3.3: Privately negotiated trade, two-party report.

The counterparty is referenced by the marketplace assigned member code in *PartyID* and optionally by the trader group in *PartySubID* (*PartySubIDType* = System). The trader group is required for manual trade reports sent to the exchange. In addition, for manual trades, traders can specify a trader id (free text) in *PartySubID* (*PartySubIDType* = Person) for both the own side and the counterparty.

In general the following trade messages are sent from the market place.

New automatically matched trade from marketplace.	**Trade Capture Report** TradeReportTransType = New (0) TradeReportType = Submit (0) TradeHandlingInstr = Trade Confirm ('0') TradeReportID=<new> TradeID=<reference> MatchStatus = Affirmed ('0')
Cancel trade from marketplace.	**Trade Capture Report** TradeReportTransType = Cancel (1) TradeReportType = Trade Report Cancel (6) TradeHandlingInstr = Trade Confirm ('0') TradeReportID=<new> TradeReportRefID=<marketplace's> TradeID=<reference> MatchStatus = Affirmed ('0')

3.9.1 One-Party Report for Pass-Thru

In the one-party report for pass-thru model the marketplace will respond each Trade Capture Report with a Trade Capture Report Ack. The messages are filled in as follows in each step of this model.

Initiator submit to marketplace.	**Trade Capture Report** TradeReportTransType = New (0) TradeReportType = Submit (0) TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportID=<new>

Ack from marketplace of initiator submit.	**Trade Capture Report Ack** TradeReportTransType = New (0) TradeReportType = Submit (0) TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportID=<initiator's>
Marketplace forward of submit to counterparty.	**Trade Capture Report** TradeReportTransType = New (0) TradeReportType = Alleged (1) TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportID=<new> MatchStatus = Unaffirmed ('1')
Inititator cancel to marketplace, before counterparty has accepted/declined.	**Trade Capture Report** TradeReportTransType = Cancel (1) TradeReportType = Submit (0) TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportRefID=<initiator's previous> TradeReportID=<new>
Ack from marketplace of inititator cancel.	**Trade Capture Report Ack** TradeReportTransType = Cancel (0) TradeReportType = Submit (0) TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportRefID=<initiator's> TradeReportID=<initiator's>
Marketplace forward of cancel to counterparty.	**Trade Capture Report** TradeReportTransType = Cancel (1) TradeReportType = Alleged (1) TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportRefID=<marketplace's> TradeReportID=<new> MatchStatus = Unaffirmed ('1')
Counterparty accept/decline to marketplace.	**Trade Capture Report** TradeReportTransType = Replace (2) TradeReportType = Accept (2) or Decline (3) TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportRefID=<marketplace's> TradeReportID=<new>
Ack from marketplace of counterparty accept/decline.	**Trade Capture Report Ack** TradeReportTransType = Replace (2) TradeReportType = Accept (2) or Decline (3) TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportRefID=<marketplace's> TradeReportID=<counterparty's>

Marketplace forward of decline to initiator.	**Trade Capture Report** TradeReportTransType = Cancel (1) TradeReportType = Decline (3) TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportRefID=<initiator's> TradeReportID=<new> MatchStatus = Unaffirmed ('1')
Marketplace confirm trade to initiator/counterparty.	**Trade Capture Report** TradeReportTransType = Replace (2) TradeReportType = Submit (0) TradeHandlingInstr = Trade Confirm ('0') TradeReportRefID=<initiator's> or <counterparty's> TradeReportID=<new> TradeID=<reference> MatchStatus = Affirmed ('0')
Reject from marketplace in response a malformed Trade Capture Report.	**Trade Capture Report Ack** TradeReportTransType = <same> TradeReportType = <same> TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportRefID=<same> TradeReportID=<same> TradeReportRejectReason=<specified>
Cancel from marketplace (due to timeout or cleanup) to initiator/counterparty.	**Trade Capture Report** TradeReportTransType = Cancel (1) TradeReportType = Alleged (1) TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportRefID=<initiator's> or <marketplace's> TradeReportID=<new> MatchStatus = Unaffirmed ('1')

3.9.2 Two-Party Report

In the two-party report model *no* Trade Capture Report Ack message is sent in response to a successful request. Instead the confirmed trade is sent directly. The fields are used in the following way in this model.

Initiator submit to marketplace.	**Trade Capture Report** TradeReportTransType = New (0) TradeReportType = Submit (0) TradeHandlingInstr = Two-Party Report ('1') TradeReportID=<new>

Marketplace confirm trade to initiator.	**Trade Capture Report** TradeReportTransType = Replace (2) TradeReportType = Submit (0) TradeHandlingInstr = Trade Confirm ('0') TradeReportRefID=<initiator's> TradeReportID=<new> TradeID=<reference> MatchStatus = Affirmed ('0')
Marketplace confirm trade to counterparty.	**Trade Capture Report** TradeReportTransType = New (0) TradeReportType = Submit (0) TradeHandlingInstr = Trade Confirm ('0') TradeReportID=<new> TradeID=<reference> MatchStatus = Affirmed ('0')
Reject from marketplace in response a malformed Trade Capture Report.	**Trade Capture Report Ack** TradeReportTransType = <same> TradeReportType = <same> TradeHandlingInstr = Two-Party Report ('1') TradeReportRefID=<same> TradeReportID=<same> TradeReportRejectReason=<specified>

3.9.3 Trade Component Block

This component block is used to define a trade.

Table 3.18: Trade.

Tag	*Field Name*	*Type*	*Req*	*Description*
1003	TradeID	string	N	Assigned by the marketplace when it records a confirmed trade.
487	TradeReportTransType	uInt32	N	Transaction type. 0=New 1=Cancel 2=Replace 3=Release 4=Reverse 5=Cancel Due To Back Out of Trade
856	TradeReportType	uInt32	N	0=Submit 1=Alleged 2=Accept 3=Decline 6=Trade Report Cancel

Table 3.18: Trade.

Tag	*Field Name*	*Type*	*Req*	*Description*
828	TrdType	uInt32	N	0=Regular Trade 52=Exchange Granted Trade
855	SecondaryTrdType	uInt32	N	Absence means '0'. Applies only to manual trades. MiFID II regulatory field. 0=Regular Trade. 64=Benchmark Trade.
1839	TrdPriceCondition	uInt32	N	Applies only to manual trades. MiFID II regulatory field. 13=Special dividend Trade. 15=Non-price forming Trade. 16=Trade not contributing to the price discovery process
1115	OrdCategory	uInt32	N	Applies only to manual trades. MiFID II regulatory field. 3=Privately Negotiated Trade
2668	TrdRegPublications	sequence	N	Applies only to manual trades. MiFID II regulatory field.
2669	→TrdRegPublication-Type	uInt32	N	0=Pre-trade transparency waiver

Table 3.18: Trade.

Tag	*Field Name*	*Type*	*Req*	*Description*
2670	→TrdRegPublReason	uInt32	N	0=No preceding order in book as transaction price set within average spread of a liquid instrument. ESMA RTS "NLIQ". 1=No preceding order in book as transaction price depends on system-set reference price for an illiquid instrument. ESMA RTS "OILQ". 2=No preceding order in book as transaction price is for transaction subject to conditions other than current market price. ESMA RTS "PRIC".
1123	TradeHandlingInstr	uInt32	N	ASCII char enumeration. '0'=Trade Confirmation '1'=Two-Party Report '3'=One-Party Report for Pass Through
32	LastQty	decimal	C	Trade quantity of this (last) fill. Required in TradeCaptureReport.
31	LastPx	decimal	C	Trade price of this (last) fill. Required in TradeCaptureReport.
60	TransactTime	uInt64	N	UTC timestamp this transaction occured. Execution time of trade or cancellation.
483	TransBkdTime	uInt64	N	UTC timestamp this trade was booked, if other than TransactTime. Used for manual trade reports and for trade cancellations. Field added.

Table 3.18: Trade.

Tag	*Field Name*	*Type*	*Req*	*Description*
573	MatchStatus	uInt32	N	The status of this trade with respect to matching or comparison. ASCII char enumeration. '0'=Compared, matched or affirmed '1'=Uncompared, unmatched, or unaffirmed
574	MatchType	uInt32	N	ASCII char enumeration. '1'=One-Party Trade Report (privately negotiated trade) '2'=Two-Party Trade Report (privately negotiated trade) '4'=Auto-match '7'=Call Auction 'x'=Manually Matched Trade Report

Table 3.18: Trade.

Tag	*Field Name*	*Type*	*Req*	*Description*
277	TradeCondition	string	N	Trade conditions set by exchange. Multiple char value (delimited with space). Field added. I=Sold Last (late reporting) AV=Outside Spread X0=Outside Spread Unknown XB=Knockout buyback Trade XS=Buyback Trade XD=Distribution Trade XAO=Opening auction Trade XAC=Closing auction Trade XAD=Circuit breaker dynamic auction Trade XAS=Circuit breaker static auction Trade XAP=Order protection auction Trade XAR=Missing reference price auction trade XLI=Large In Scale trade
552	Sides	sequence	C	Conditionally required in TradeCaptureReport when TradeID is not [N/A].
54	→Side	uInt32	Y	ASCII char enumeration. '1'=buy '2'=sell
37	→OrderID	string	N	
20028	→OrderPriority	uInt64	N	Indicates the priority of the order in the order-book in comparison to other orders on the same level. Higher value means lower priority. Custom field.

Table 3.18: Trade.

Tag	*Field Name*	*Type*	*Req*	*Description*
11	→ClOrdID	string	N	Client assigned order id in case of an order. In the case of quotes mapped to QuoteMsgID of a single Quote.
526	→SecondaryClOrdID	string	N	In the case of quotes mapped to QuoteID of a single Quote.
1	→Account	string	N	Account as specified in the order or Trade Capture Request.
1093	→LotType	uInt32	N	Defines the lot type assigned to the order. ASCII char enumeration. '1'=Odd Lot '2'=Round Lot
1057	→AggressorIndicator	uInt32	N	Used to identify whether the order initiator is an aggressor or not in the trade. Boolean. 'Y'=Order initiator is aggressor 'N'=Order initiator is passive
528	→OrderCapacity	uInt32	N	Designates the capacity of the firm placing the order. Absence means 'R' for trades reported to the exchange. 'P'=Principal (Deal) 'R'=Riskless principal (Matched) 'A'=Agency (Any other capacity)
529	→OrderRestrictions	string	N	Multiple char value (delimited with space). Restrictions associated with an order. B=Issuer Holding C=Issue Price Stabilization

Table 3.18: Trade.

Tag	*Field Name*	*Type*	*Req*	*Description*
159	→AccruedInterestAmt	decimal	N	Amount of accrued interest the buyer compensates the seller. Applicable for bonds and fixed income.
1724	→OrderOrigination	uInt32	N	Identifies the origin of the order. Absence means non DEA. '5'=Order received from a direct access or sponsored access customer
	→ **component block <Parties>**			
	→ **component block <OrderAttributeGrp>**			

3.9.4 Trade Capture Report (AE)

The *Trade Capture Report* message is used by the exchange to send confirmed trades. It is also used in manual trade reporting.

TradeCaptureReport:

- is replied to with a *TradeCaptureReport* message, with TradeReportRefID set to (copied from) the value in the request message
- is replied to with a *TradeCaptureReportAck* message, with TradeReportRejectReason set to 0 (Successful) and TradeReportID set to (copied from) the value in the request message
- can be rejected with a *TradeCaptureReportAck* message, with TradeReportRejectReason set to 1 (InvalidPartyInformation), 2 (UnknownInstrument), 3 (UnauthorizedToReportTrades), 4 (InvalidTradeType), 99 (Other) or 100 (ManualTradesNotAllowedInAnyKnockoutState) and TradeReportID set to (copied from) the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to AE
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the TradeCaptureReport message

TradeCaptureReport is sent:

- unsolicited, when a trade occurs

- in reply to a *TradeCaptureReport* message, with TradeReportRefID set to (copied from) the value in the request message
- in reply to a *TradeCaptureReportRequest* message, with TradeRequestID set to (copied from) the value in the request message

Table 3.19: TradeCaptureReport (AE).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
571	TradeReportID	string	N	Assigned by the submitter of the message and used as a pure message identifier.
572	TradeReportRefID	string	N	The TradeReportID that is being referenced for some action, such as correction or cancelation.
568	TradeRequestID	string	N	Request ID if this message is in response to a Trade Capture Report Request.
912	LastRptRequested	uInt32	N	Indicates that this is the last report which will be returned as a result of the request. ASCII char enumeration (boolean). 'N'=Not Last Message 'Y'=Last Message
	component block <SecurityRef>			
	component block <Trade>			

3.9.5 Trade Capture Report Ack (AR)

The *Trade Capture Report Ack* message is used for rejects. It is also used to acknowledge receival of trade capture reports in the following cases:

- Initiator's trade capture report (both new and cancel) for a one-party report for pass through.
- Counterparty's decline of a one-party report for pass through.

In other cases the confirmed trade capture report can be seen as an acknowledgement. This means that the *Trade Capture Report* will always be directly replied to with a message.

TradeCaptureReportAck is sent:

- in reply to a *TradeCaptureReport* message, with TradeReportRejectReason set to 0 (Successful) and TradeReportID set to (copied from) the value in the request message
- to reject a *TradeCaptureReport* message, with TradeReportRejectReason set to 1 (InvalidPartyInformation), 2 (UnknownInstrument), 3 (UnauthorizedToReportTrades), 4 (InvalidTradeType), 99 (Other) or 100 (ManualTradesNotAllowedInAnyKnockoutState) and TradeReportID set to (copied from) the value in the request message

Table 3.20: TradeCaptureReportAck (AR).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
571	TradeReportID	string	N	Assigned by the submitter of the message and used as a pure message identifier.
572	TradeReportRefID	string	N	The TradeReportID that is being referenced for some action, such as correction or cancelation.
568	TradeRequestID	string	N	Request ID if this message is in response to a Trade Capture Report Request.
912	LastRptRequested	uInt32	N	Indicates that this is the last report which will be returned as a result of the request. ASCII char enumeration (boolean). 'N'=Not Last Message 'Y'=Last Message

Table 3.20: TradeCaptureReportAck (AR).

Tag	*Field Name*	*Type*	*Req*	*Description*
751	TradeReportReject-Reason	uInt32	N	0=Successful (default) 1=Invalid party information 2=Unknown instrument 3=Unauthorized to report trades 4=Invalid trade type 5=Manual trades are not allowed for this instrument 6=Manual trades that add to DVC limits not allowed for this instrument. 7=Trade for this specific instrument and/or member is blocked by a killswitch. 99=Other 100=Manual trades not allowed in any knockout state
	component block <SecurityRef>			
	component block <Trade>			
58	Text	string	N	Error message.

3.9.6 Trade Capture Report Request (AD)

All trade capture reports involving the requester's trader group can be requested with the *Trade Capture Report Request* message with *TradeRequestType* set to 0 (All Trades). Only trades for the last 72 hours are available. The time interval can be narrowed further by setting *TradeRequestType* to 1 and specifying the time interval in the *Dates* sequence. This message will be replied to with one or more *Trade Capture Report* messages. The last *Trade Capture Report* will be indicated with *LastRptRequested* field set to 'Y'. Note that a dummy *Trade Capture Report* with *TradeID* set to "[N/A]" and *LastRptRequested* field set to 'Y' may be sent as last message to indicate the request has been processed (for example as a response with no trades).

In the event of a malformed request, the response will be a *Trade Capture Report Request Ack* message.

TradeCaptureReportRequest:

- is replied to with a *TradeCaptureReport* message, with TradeRequestID set to (copied from) the value in the request message
- is replied to with a *TradeCaptureReportRequestAck* message, with TradeRequestResult set to 0 (Successful) and TradeRequestID set to (copied from) the value in the request message
- can be rejected with a *TradeCaptureReportRequestAck* message, with TradeRequestResult set to 1 (InvalidOrUnknownInstrument), 2 (InvalidTypeOrTradeRequested), 3 (InvalidParties), 4 (InvalidTransportTypeRequested), 5 (InvalidDestinationRequest), 8 (TradeRequestTypeNotSupported), 9 (NotAuthorized) or 99 (Other) and TradeRequestID set to (copied from) the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to AD
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the TradeCaptureReportRequest message

Table 3.21: TradeCaptureReportRequest (AD).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
568	TradeRequestID	string	Y	Identifier for the trade request.
569	TradeRequestType	uInt32	Y	0=All trades (last 48 hours) 1=Matched trades matching criteria provided on request
580	Dates	sequence	N	Range of dates. *Since* (NoDates=1) or *Between* (NoDates=2) dates, inclusive.
60	→TransactTime	uInt64	Y	UTC time the trade was created.

3.9.7 Trade Capture Report Request Ack (AQ)

This message is only sent as a reject to a *Trade Capture Report Request*.

TradeCaptureReportRequestAck is sent:

- in reply to a *TradeCaptureReportRequest* message, with TradeRequestResult set to 0 (Successful) and TradeRequestID set to (copied from) the

value in the request message

- to reject a *TradeCaptureReportRequest* message, with TradeRequestResult set to 1 (InvalidOrUnknownInstrument), 2 (InvalidTypeOrTradeRequested), 3 (InvalidParties), 4 (InvalidTransportTypeRequested), 5 (InvalidDestinationRequest), 8 (TradeRequestTypeNotSupported), 9 (NotAuthorized) or 99 (Other) and TradeRequestID set to (copied from) the value in the request message

Table 3.22: TradeCaptureReportRequestAck (AQ).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
568	TradeRequestID	string	Y	Identifier for the trade request.
569	TradeRequestType	uInt32	Y	0=All trades (last 48 hours) 1=Matched trades matching criteria provided on request
749	TradeRequestResult	uInt32	Y	Result of Trade Request. 0=Successful (default) 1=Invalid or unknown instrument 2=Invalid type of trade requested 3=Invalid parties 4=Invalid transport type requested 5=Invalid destination requested 8=TradeRequestType not supported 9=Not authorized 99=Other
750	TradeRequestStatus	uInt32	Y	Status of Trade Request. 0=Accepted 1=Completed 2=Rejected
58	Text	string	N	Error message.

3.10 Financial Status Messages

3.10.1 User Security Status Update Request (FU)

The *User Security Status Update Request* message allows a member with sufficient rights to change the financial status of a specific instrument. If the request is accepted, the new financial status will be published by a *Security Status* message on the public channel.

A request to knock the instrument will be replied with the status being changed to *Knock out* or *Knock out buyback*. The latter will be replied if the instrument is registered as a *Buy-Back* instrument.

UserSecurityStatusUpdateRequest:

- is replied to with an *UserSecurityStatusUpdateResponse* message, with SecurityStatusUpdateRequestID set to (copied from) the value in the request message
- can be rejected with an *UserSecurityStatusUpdateResponse* message, with FinancialStatusResult set to 1 (UnknownSecurityId), 2 (InvalidFinancialStatus), 3 (InsufficientRigths) or 4 (Other) and SecurityStatusUpdateRequestID set to (copied from) the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to FU
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the UserSecurityStatusUpdateRequest message

Table 3.23: UserSecurityStatusUpdateRequest (FU).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
	component block <SecurityRef>			
20040	SecurityStatusUpdate-RequestID	string	Y	
20049	FinancialStatus-Updates	sequence	N	
20038	→FinancialStatus-UpdateType	uInt32	Y	Financial status type. 1=Knock instrument (will result in knockout or knockout buyback) 2=Soft-knock instrument 3=Buyback 4=Distribution 5=Protection Mode(Trigger auction when Market Maker quote is missing)

Table 3.23: UserSecurityStatusUpdateRequest (FU).

Tag	*Field Name*	*Type*	*Req*	*Description*
20050	→FinancialStatus-UpdateValue	uInt32	Y	Financial status operation. 1=Enable 2=Clear

3.10.2 User Security Status Update Response (FR)

UserSecurityStatusUpdateResponse is sent:

- in reply to an *UserSecurityStatusUpdateRequest* message, with SecurityStatusUpdateRequestID set to (copied from) the value in the request message
- to reject an *UserSecurityStatusUpdateRequest* message, with FinancialStatusResult set to 1 (UnknownSecurityId), 2 (InvalidFinancialStatus), 3 (InsufficientRigths) or 4 (Other) and SecurityStatusUpdateRequestID set to (copied from) the value in the request message

Table 3.24: UserSecurityStatusUpdateResponse (FR).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
	component block <SecurityRef>			
20040	SecurityStatusUpdate-RequestID	string	Y	

Table 3.24: UserSecurityStatusUpdateResponse (FR).

Tag	*Field Name*	*Type*	*Req*	*Description*
291	FinancialStatus	string	N	Multiple char value (delimited with space). All values are mutually exclusive except 'Under observation' and 'Order protection mode' which can appear together with any of the others. W=Knockout X=Knockout buyback Y=Knockout soft U=Buyback V=Distribution Z=Under observation D=Circuit breaker dynamic S=Circuit breaker static M=Order protection mode P=Order protection auction Q=Manual matching C=Recalculated
20042	FinancialStatusResult	uInt32	Y	Financial status update result. 0=Success 1=Unknown Security ID 2=Unsupported financial operation 3=User does not have sufficient rights to update financial status 4=Other error
58	Text	string	N	Message to explain reason in case of rejection

Chapter 4

Public Service

The public service is mainly used for receiving reference data and market data from the exchange. The traffic is almost entirely of a non-interactive "broadcast" nature. Non-interactive since information is sent spontaneously from the exchange (not in direct response to a request from the user). Broadcast since the same information is sent to all users of the service.

Examples of non-interactive traffic include public orders and trades as well as security definitions. An example of interactive traffic is snapshot messages.

As a consequence of the non-interactive and broadcast properties of the service, data (typically orders from other users) is pushed to a user's session even when a user is offline. No subscription requests are required nor supported by the service. Instead, a user needs to synchronize with the service when logging on, either on the session level (by requesting retransmission of lost messages) or on the application level (by requesting snapshots).

Note that for scalability reasons the public service can be divided into multiple FIX sessions. The public data is then partitioned by security, meaning that security data and market data for a given security is only sent on one of the FIX sessions. Reference data such as market structure and trading session status is sent on all FIX sessions.

When multiple FIX sessions are used, the sessions should be considered independent of each other since no guarantees regarding timing between the sessions can be made.

4.1 Full Snapshot Recovery

On the public service snapshots can be requested for the following:

Market Structure See the *Market Definition Request* message in section 4.5.2.

Trading Session Status See the *Trading Session Status Request* message in section 4.5.6.

Securities See the *Security List Request* message in section 4.4.2.

Security Status See the *Security Mass Status Request* message in section 4.4.5.

Market Data See the *Market Data Request* message in section 4.6.2.

Corporate Actions See the *Corporate Action Request* message in section 4.7.3.

4.2 Message Overview

The following messages can be sent/received by the client to/from the public service. Depending on the filter rules only a subset of the following messages may be sent/received.

Note that since no operations that modify data are permitted on the public service the messages for *All* and *Read-only* filtering rules are the same.

Table 4.1: Message overview.

Message	*Class*	*All?*	*Read-only?*	*Snapshot-only?*
MarketDataRequest	Market data	send	send	send
MarketDataSnapshotFullRefresh	Market data	recv	recv	recv
MarketDataIncrementalRefresh	Market data	recv	recv	
MarketDataRequestReject	Market data	recv	recv	recv
SecurityListRequest	Security	send	send	send
SecurityList	Security	recv	recv	recv
SecurityDefinitionUpdateReport	Security	recv	recv	
SecurityMassStatusRequest	Security status	send	send	send
SecurityStatus	Security status	recv	recv	recv
MarketDefinitionRequest	Market structure	send	send	send
MarketDefinition	Market structure	recv	recv	recv
MarketDefinitionUpdateReport	Market structure	recv	recv	
TradingSessionStatusRequest	Trading session status	send	send	send
TradingSessionStatus	Trading session status	recv	recv	recv
CorporateActionReport	Corporate action	recv	recv	recv
CorporateActionRequest	Corporate action	send	send	send

4.2.1 Filtering Examples

The following are examples of filter rules that can be useful when not all information is required or can be handled.

Reference data is only needed, i.e. list of securities and market segments:
Market Structure=read-only, Trading Session Status=none, Securities=read-only, Security Status=none, Market Data=none, Corporate Actions=none.

Reference data and corporate actions is needed, i.e. list of securities and market segments plus information about corporate actions (splits etc.):
Market Structure=read-only, Trading Session Status=none, Securities=read-only, Security Status=none, Market Data=none, Corporate Actions=read-only.

Reference data with status is needed, i.e. list of securities and market segments and the trading status of the market segments and securities:
Market Structure=read-only, Trading Session Status=read-only, Securities=read-only, Security Status=read-only, Market Data=none, Corporate Actions=none.

4.3 Component Blocks

4.3.1 Security Defaults

Security parameters that can have default values on the market segment level, and overridden on security level.

Table 4.2: SecurityDefaults.

Tag	*Field Name*	*Type*	*Req*	*Description*
15	Currency	string	N	ISO 4217 currency code.
543	InstrRegistry	string	N	Values may include BIC for the depository or custodian who maintain ownership records, the ISO country code for the location of the record, or the value "ZZ" to specify physical ownership of the security (e.g. stock certificate).
40471	BusinessCenter	string	N	A business center whose calendar is used for date adjustment, e.g. "GBLO".
20070	ZoneID	string	N	The IANA Time Zone identifier which is used for local time and date conversions. Custom field.

4.3.2 Trading Rules

Trading rules that can be specified on market segment level and overridden on security level.

Table 4.3: TradingRules.

Tag	*Field Name*	*Type*	*Req*	*Description*
562	MinTradeVol	decimal	N	Minimum trading volume that can be submitted
561	RoundLot	decimal	N	
423	PriceType	uInt32	N	Defines the default Price Type used for trading. 1=Percentage (i.e. percent of par) 2=Per unit (i.e. per share or contract)
20054	MaxOrderExpire-Duration	uInt32	N	Max duration in seconds of ExpireTime in GTC orders. Custom field.
20055	MaxTradeTransBkd-TimeDiff	uInt32	N	Max time difference in seconds between Trans-actTime and TransBkd-Time of trades, i.e. how far back in time a man-ual trade can be reported. Custom field.
1205	TickRules	sequence	N	This block specifies the rules for determining how a security ticks, i.e. the price increments at which it can be quoted and traded.
1206	→StartTickPrice-Range	decimal	N	Starting price range for specified tick increment.
1207	→EndTickPriceRange	decimal	N	Ending price range for specified tick increment.
1208	→TickIncrement	decimal	N	Tick increment for stated price range.
1235	MatchRules	sequence	N	

Table 4.3: TradingRules.

Tag	*Field Name*	*Type*	*Req*	*Description*
1142	→MatchAlgorithm	string	Y	The type of algorithm used to match orders in this market segment. price-time=FIFO matching with price-time order priority. price-internal-time=FIFO matching with price-internal-time order priority.
574	→MatchType	uInt32	N	The point in the matching process at which the matching algorithm applies. ASCII char enumeration. '4'=Auto-match (continuous trading) '7'=Call Auction 'x'=Manually Matched Trade Report
20056	MarketOrderRules	sequence	N	The rules that applies for market order. Custom field.

Table 4.3: TradingRules.

Tag	*Field Name*	*Type*	*Req*	*Description*
20057	→MarketOrderRule	uInt32	Y	The market order rules that applies. Custom field. 1=Allow instantenous (IOC or FoK) market orders and during auctions. 2=Allow market orders to be placed into the order book. 3=Market order protection enabled. Indicates whether retailers are ensured that the market maker is present when submitting instantenous (IOC or FoK) market orders. Furthermore it allows the instrument to enter 'Order Protection Mode'. 4=Reveal market order in market data. 5=Match immediate market order only against the best price level during continuous trading. Not applicable to non-immediate market orders.
20058	OrderProtection-AuctionTimeMin	uInt32	N	Lower bound in milliseconds of duration of the order protection auction.Custom field.
20059	OrderProtection-AuctionTimeMax	uInt32	N	Upper bound in milliseconds of duration of the order protection auction.Custom field.
20067	MissingReference-PriceAuctionTimeMin	uInt32	N	Lower bound in milliseconds of duration of the missing reference price auction.Custom field.

Table 4.3: TradingRules.

Tag	*Field Name*	*Type*	*Req*	*Description*
20068	MissingReference-PriceAuctionTimeMax	uInt32	N	Upper bound in milliseconds of duration of the missing reference price auction.Custom field.
20052	AllowReserveOrder	uInt32	N	Indicates whether reserve orders are allowed on this instrument. ASCII char enumeration (boolean). Custom field. 'Y'=Reserve order allowed on instrument 'N'=Reserve order not allowed on instrument
20051	MinReserveOrder-Value	decimal	N	Minimum reserve order value, applicable for both new orders and order modifications. If the field is absent or set to 0 it means that there are no minimum value. Custom field.
20060	MinReserveOrder-ValueCurrency	string	N	Currency for MinReserveOrderValue. ISO 4217 currency code. Custom field.
20061	MarketDataRules	sequence	N	Market data visibility rules. Custom field.
20062	→MarketDataRule	uInt32	Y	Custom field. 1=Reveal counterparty information for orders and trades 2=Distribute orders during Pre-Open 3=Distribute equilibrium price during auctions
20063	PartyRules	sequence	N	Party information rules that applies. Custom field.

Table 4.3: TradingRules.

Tag	*Field Name*	*Type*	*Req*	*Description*
20064	→PartyRule	uInt32	N	Indicates which PartyID information must be present. 1=Executing trader is required for orders and quotes. 2=ClientID is required for orders. 3=ClientID is NOT permitted for quotes.
20065	TradeReportRules	sequence	N	Rules for manual trade reports. Custom field.
20066	→TradeReportRule	uInt32	N	Custom field. 1=Allow all trade reports. 2=Allow only trade reports that do not add to the Double Volume Cap (DVC) limits.

4.4 Security Messages

In this document *order book* and *security* are used interchangeably. Two order books for the same instrument (e.g. different currencies) will be defined as two securities.

4.4.1 Security Component Block

This component block is used to define a security. The security is described in detail using the *SecurityXML* field. The format of the XML is described in *NGM XML Security Specification*.

The *PriceType* of the security controls the type of the *Price* field in orders and quotes for the security. When *PriceType* is percentage then a price of `99.5%` is is specified as `Price=99.5`.

Table 4.4: Security.

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <SecurityRef>			
454	SecurityAltIDs	sequence	N	

Table 4.4: Security.

Tag	*Field Name*	*Type*	*Req*	*Description*
455	→SecurityAltID	string	Y	Alternative security identifier of type specified in SecurityAltIDSource.

Table 4.4: Security.

Tag	*Field Name*	*Type*	*Req*	*Description*
456	→SecurityAltIDSource	uInt32	Y	Identifies the class of SecurityID. ASCII char enumeration. '1'=CUSIP '2'=SEDOL '3'=QUIK '4'=ISIN '5'=RIC code '6'=ISO Currency Code '7'=ISO Country Code '8'=Exchange Symbol '9'=Consolidated Tape Association (CTA) Symbol (SIAC CTS/CQS line format) 'B'=Wertpapier 'C'=Dutch 'D'=Valoren 'E'=Sicovam 'F'=Belgian 'G'="Common" (Clearstream and Euroclear) 'H'=Clearing House / Clearing Organization 'J'=Option Price Reporting Authority 'L'=Letter of Credit 'M'=Marketplace-assigned identifier 'N'=Markit RED Entity CLIP 'P'=Markit RED Pair CLIP 'Q'=CFTC Commodity Code 'R'=ISDA Commodity Reference Price 'S'=Financial Instrument Global Identifier 'T'=Legal Entity Identifier 'U'=Synthetic 'x'=Secondary market-place-assigned identifier

Table 4.4: Security.

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <SecurityDefaults>			
1310	MarketSegments	sequence	N	A security is strictly member of one market segment.
1301	→MarketID	string	N	Identifies the market. ISO 10383 Market Identifier Code (MIC).
1300	→MarketSegmentID	string	N	Identifies the market segment.
	→ component block <TradingRules>			
1185	SecurityXML	string	N	XML data describing the security.
20069	LiquidityStatus	uInt32	N	Liquidity status classification of this security. Absence means unknown or N/A.Custom field. 1=Liquid 2=Illiquid

4.4.2 Security List Request (x)

A list of the all available securities are requested with the *Security List Request* message. The request will be replied to with one or more *Security List* messages. The last *Security List* message will always be indicated with the *LastFragment* field set to 'Y'. Note that a reply with 0 repeating securities may be sent as a reply.

In the event of a malformed request, the response will be a *Security List* message with *SecurityRequestResult* set to 1 (Invalid or unsupported request).

SecurityListRequest:

- is replied to with a *SecurityList* message, with SecurityRequestResult set to 0 (ValidRequest) and SecurityReqID set to (copied from) the value in the request message
- can be rejected with a *SecurityList* message, with SecurityRequestResult set to 1 (InvalidOrUnsupportedRequest) and SecurityReqID set to (copied from) the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to x

- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the SecurityListRequest message

Table 4.5: SecurityListRequest (x).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
320	SecurityReqID	string	Y	

4.4.3 Security List (y)

Response to *Security List Request*.

SecurityList is sent:

- in reply to a *SecurityListRequest* message, with SecurityRequestResult set to 0 (ValidRequest) and SecurityReqID set to (copied from) the value in the request message
- to reject a *SecurityListRequest* message, with SecurityRequestResult set to 1 (InvalidOrUnsupportedRequest) and SecurityReqID set to (copied from) the value in the request message

Table 4.6: SecurityList (y).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
320	SecurityReqID	string	N	
560	SecurityRequestResult	uInt32	N	0=Valid request (default) 1=Invalid or unsupported request
893	LastFragment	uInt32	N	Indicates whether this is the last fragment in a sequence of message fragments. ASCII char enumeration (boolean). 'N'=Not Last Message 'Y'=Last Message
146	RelatedSym	sequence	N	
	→ **component block <Security>**			

4.4.4 Security Definition Update Report (BP)

Incremental (unsolicited) update of available securities.

SecurityDefinitionUpdateReport is sent:

- unsolicited, when a change occurs

Table 4.7: SecurityDefinitionUpdateReport (BP).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
980	SecurityUpdateAction	uInt32	N	ASCII char enumeration. 'A'=Add 'D'=Delete 'M'=Modify
20027	SecurityMoveIndicator	uInt32	N	ASCII char enumeration. Abscence means No 'Y'=Yes. The SecurityUpdateAction (Add/Delete) is a move between two market data channels. 'N'=No. The security appears for the first time/is permanently removed
	component block <Security>			
58	Text	string	N	Comment, instructions or other identifying information.

4.4.5 Security Mass Status Request (NGM-ex)

The status of all securities can be requested with the *Security Mass Status Request* message. The reply is one or more *Security Status* messages. The last *Security Status* message will always be indicated with the *LastRptRequested* field set to 'Y'. In the unlikely event that there is no security defined a dummy *Security Status* message with *SecurityID* absent (null) and *LastRptRequested* field set to 'Y' will be sent as a response.

Notice that the security status snapshot and the security list snapshot is an

exception that all replies are in the same order as the requests sent. The correct behaviour to counter this is to request the security status once the complete security list has been received.

If no *Security Status* message is received for a security the trading status should be considered closed.

SecurityMassStatusRequest:

- is replied to with a *SecurityStatus* message, with SecurityStatusReqID set to (copied from) the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to NGM-ex
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the SecurityMassStatusRequest message

Table 4.8: SecurityMassStatusRequest (NGM-ex).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
324	SecurityStatusReqID	string	Y	

4.4.6 Security Stat Component Block

This component block is used to describe the status of a security.

Table 4.9: SecurityStat.

Tag	*Field Name*	*Type*	*Req*	*Description*
336	TradingSessionID	string	N	Identifier for trading session. 1=Day (regular session) 6=After-hours (non-regular session)

Table 4.9: SecurityStat.

Tag	*Field Name*	*Type*	*Req*	*Description*
326	SecurityTradingStatus	uInt32	N	2=Trading halt 4=No Open / No Resume (closed) 17=Ready to trade (open) 18=Not available for trading (post open) 20=Unknown or Invalid (Request Reject) 21=Pre-open 101=Opening auction 102=Closing auction
327	HaltReason	uInt32	C	Conditionally required when SecurityTradingStatus is 2 (Trading halt). Denotes the reason for the Opening Delay or Trading Halt. ASCII char enumeration. 'R'=Regulatory Halt 'O'=Other
328	InViewOfCommon	uInt32	N	Indicates whether or not the halt was due to common stock trading being halted. ASCII char enumeration (boolean). 'N'=Halt was not related to a halt of the common stock 'Y'=Halt was due to common stock being halted
329	DueToRelated	uInt32	N	Indicates whether or not the halt was due to the related security being halted. ASCII char enumeration (boolean). 'N'=Halt was not related to a halt of the related security 'Y'=Halt was due to related security being halted

Table 4.9: SecurityStat.

Tag	*Field Name*	*Type*	*Req*	*Description*
292	CorporateAction	string	N	Multiple char value (delimited with space). A=Ex-Dividend C=Ex-Rights I=Reverse Stock Split J=Standard-Integer Stock Split Q=Tender Offer
291	FinancialStatus	string	N	Multiple char value (delimited with space). All values are mutually exclusive except 'Under observation' and 'Order protection mode' which can appear together with any of the others. W=Knockout X=Knockout buyback Y=Knockout soft U=Buyback V=Distribution Z=Under observation D=Circuit breaker dynamic S=Circuit breaker static M=Order protection mode P=Order protection auction R=Missing reference price auction Q=Manual matching C=Recalculated

4.4.7 Security Status (f)

The *Security Status* message is used for unsolicited updates of security status and for replies to a *Security Mass Status Request*.

SecurityStatus is sent:

- unsolicited, when a change occurs

- in reply to a *SecurityMassStatusRequest* message, with SecurityStatusReqID set to (copied from) the value in the request message

Table 4.10: SecurityStatus (f).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
324	SecurityStatusReqID	string	N	
912	LastRptRequested	uInt32	N	Indicates that this is the last report which will be returned as a result of the request. ASCII char enumeration (boolean). Field added. 'N'=Not Last Message 'Y'=Last Message
	component block <SecurityRef>			
	component block <SecurityStat>			

4.5 Market Structure Messages

Each security belongs to one (and only one) market segment. The market segments can be organized in a hierarchy, but market segments do not inherit properties and status from their parent market segment. Each market segment has one (and only one) trading session.

The status of a trading session is conveyed using the *Trading Session Status* message, and this will affect all securities that follow the market segment (i.e. is not trade halted, etc.). The status of each security is also sent individually using the *Security Status* message. The timing between the trading session status and the security status is not perfect which means that the security status should used to see if e.g. the security is open for trading and the trading session status should be used to see if the market segment is open or not.

4.5.1 Market Component Block

This component block is used to define a market.

Table 4.11: Market.

Tag	*Field Name*	*Type*	*Req*	*Description*
1301	MarketID	string	Y	ISO 10383 Market Identifier Code (MIC).
1300	MarketSegmentID	string	N	Identifies the market segment.

Table 4.11: Market.

Tag	*Field Name*	*Type*	*Req*	*Description*
1396	MarketSegmentDesc	string	N	Description or name of market segment.
1398	EncodedMktSegmDesc	string	N	Encoded (non-ASCII) description or name of market segment.
1325	ParentMktSegmID	string	N	Reference to a parent market segment.
	component block <SecurityDefaults>			
	component block <TradingRules>			

4.5.2 Market Definition Request (BT)

A snapshot of the market structure can be obtained through a *Market Definition Request* message. The request will be replied to with one or more *Market Definition* messages. The last *Market Definition* message will always be indicated with *LastRptRequested* field set to 'Y'. In the unlikely event that there are no market or market segments defined a dummy *Market Definition* message with *MarketID* set to "[N/A]" and *LastRptRequested* field set to 'Y' will be sent as a response.

In the event of a malformed request, the response will be a *Business Message Reject* message.

MarketDefinitionRequest:

- is replied to with a *MarketDefinition* message, with MarketReqID set to (copied from) the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to BT
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the MarketDefinitionRequest message

Table 4.12: MarketDefinitionRequest (BT).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
1393	MarketReqID	string	Y	Unique request id.
263	SubscriptionRequest-Type	uInt32	Y	ASCII char enumeration. '0'=Snapshot

4.5.3 Market Definition (BU)

The *Market Definition* message is used for delivering a snapshot of the market structure.

MarketDefinition is sent:

- in reply to a *MarketDefinitionRequest* message, with MarketReqID set to (copied from) the value in the request message

Table 4.13: MarketDefinition (BU).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
1393	MarketReqID	string	N	Reference to the request.
912	LastRptRequested	uInt32	N	Indicates that this is the last report which will be returned as a result of the request. ASCII char enumeration (boolean). Field added. 'N'=Not Last Message 'Y'=Last Message
	component block <Market>			

4.5.4 Market Definition Update Report (BV)

The *Market Definition Update Report* message is used for delivering an incremental update of the market structure.

MarketDefinitionUpdateReport is sent:

- unsolicited, when a change occurs

Table 4.14: MarketDefinitionUpdateReport (BV).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
1394	MarketReportID	string	Y	Unique identifier for each MarketDefinitionUpdateReport message.

Table 4.14: MarketDefinitionUpdateReport (BV).

Tag	*Field Name*	*Type*	*Req*	*Description*
1395	MarketUpdateAction	uInt32	N	ASCII char enumeration. 'A'=Add 'D'=Delete 'M'=Modify
	component block <Market>			

4.5.5 Trading Session Component Block

This component block is used to describe the trading session status of a market.

Table 4.15: TradingSession.

Tag	*Field Name*	*Type*	*Req*	*Description*
1301	MarketID	string	N	Market for which Trading Session applies.
1300	MarketSegmentID	string	N	Market Segment for which Trading Session applies.
335	TradSesReqID	string	N	Trading Session Status Request ID
336	TradingSessionID	string	N	Identifier for trading session. 1=Day (regular session) 6=After-hours (non-regular session)
340	TradSesStatus	uInt32	Y	State of the trading session. 0=Unknown 1=Halted 2=Open 3=Closed 4=Pre-Open 5=Pre-Close 6=Request Rejected 7=Opening auction 8=Closing auction

Table 4.15: TradingSession.

Tag	*Field Name*	*Type*	*Req*	*Description*
912	LastRptRequested	uInt32	N	Indicates that this is the last message which will be returned as a result of the request. Field added. ASCII char enumeration (boolean). 'N'=Not Last Message 'Y'=Last Message
58	Text	string	N	Error message.

4.5.6 Trading Session Status Request (g)

The status of the trading sessions (market segments) can be obtained through the *Trading Session Status Request* message. The request will be replied to with one or more *Trading Session Status* messages. The last *Trading Session Status* message will always be indicated with *LastRptRequested* field set to 'Y'. In the unlikely event that there is no market or trading session (market segment) defined a dummy *Trading Session Status* message with *MarketID* set to "[N/A]" and *LastRptRequested* field set to 'Y' will be sent as a response.

TradingSessionStatusRequest:

- is replied to with a *TradingSessionStatus* message, with TradSesReqID set to (copied from) the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to g
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the TradingSessionStatusRequest message

Table 4.16: TradingSessionStatusRequest (g).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
335	TradSesReqID	string	Y	Unique request id.
263	SubscriptionRequest-Type	uInt32	Y	ASCII char enumeration. '0'=Snapshot

4.5.7 Trading Session Status (h)

Provides information on the status of a market. The *Trading Session Status* message is sent both as a reply to a previous request and unsolicited whenever the status of a trading session changes.

TradingSessionStatus is sent:

- unsolicited, when a change occurs
- in reply to a *TradingSessionStatusRequest* message, with TradSesReqID set to (copied from) the value in the request message

Table 4.17: TradingSessionStatus (h).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
	component block <TradingSession>			

4.6 Market Data Messages

The *MDEntryID* field contains the trade id for trades and the public order id for orders. The id is static, meaning that it will not change through the lifetime of the order or the trade. It is not used for other entry types (e.g. high price).

Bid ('0') *MDEntryPx* and *MDEntrySize* contains the price and volume of the bid order or quote.

Offer ('1') *MDEntryPx* and *MDEntrySize* contains the price and volume of the offer order or quote.

Trade ('2') *MDEntryPx* and *MDEntrySize* contains the price and volume of the trade.

The statistics are maintained for *session*, *day* and *official (day)*. The values can be requested in a snapshot until they are generated or cleared next time. Session can be defined as regular or non regular. Regular sessions is the regular trading and define the closing price while the non-regular sessions do not affect the closing price (typically after hours and similar). A non-regular session will have *TradingSessionID* set to "6" (After-hours), while the regular sessions will have *TradingSessionID* set to "1" (Day).

Session *MDStatScope* set to "1". The Session runs from the moment the security status enters pre-open until it is closed. If a snapshot is requested it will send the current statistics (in synchronization with incremental updates) so the client can continue calculating the statistics with trades as

a basis. If a snapshot is asked when an order book is closed, the statistics of the last session will be sent. When the statistics are reset at the start of the pre-trade an increment with all values except closing (which will be the closing of the previous regular session) set to 0 will be sent.

Day *MDStatScope* set to "2". The Day statistics start at 00:00 (market time) and ends 23:59:59:999. If a snapshot is requested it will send the current statistics (in synchronization with incremental updates) so the client can continue calculating the statistics with trades as a basis. When the statistics are reset at midnight an increment with all values except closing (which will be the closing of the previous regular session) set to 0 will be sent. Also note that the Day closing price can be set to the theoretical price of an instrument, and must thus not necessarily be a direct reflection of the trades conducted in the orderbook of the instrument.

Official Day *MDStatScope* set to "3". The Official Day statistics start at 00:00 (market time) until the regular session is closed. If a snapshot is requested it will send the last official statistics that were sent (those generated at the last regular session closing). When the statistics are updated at the closing of a regular session an increment with all values will be sent. Note that the Official Day statistics does not encompass after hours session trades that occurs after the day session has ended, nor are trade cancellations that take place after the closing resulting in a change of the stats. Also note that the Official Day closing price can be set to the theoretical price of an instrument, and must thus not necessarily be a direct reflection of the trades conducted in the orderbook of the instrument.

Opening statistics for the *day session* and *official day session* is defined as the first opening of any session and the last closing taken from a regular session. Session, day and official day values are differentiated by the *MDStatsScope* field. Figure 4.1 shows how closing prices are passed on to the next trading session as reference prices.

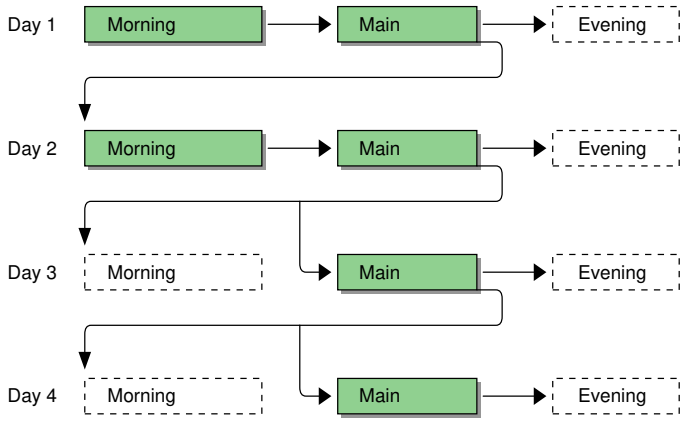

Figure 4.1: Green sessions are regular, white (and dashed) sessions are non-regular (after-hours).

Opening and closing prices are sent whenever they are cleared or generated. The closing price is generated when a regular session is closed. If no trade was made during the session and no closing price was set by the market maker organization, the previous closing price is sent. The session closing price can be adjusted for corporate actions and trade cancellations in which case it will be sent and receive a new timestamp.

Opening Price ('4') *MDEntryPx* contains the price.

Closing Price ('5') *MDEntryPx* contains the price. The *TransactTime* contains the time the closing price was generated. A day or official day closing price with the MarketMakerQuote field set to 'Y' indicates that the closing price is theoretical and based on the quotation of the market maker.

The following *MDEntryTypes* will only sent when they are reset (beginning of trading session or day) and whenever they are changed due to a trade cancellation. If the receiver need these values continuously they can be calculated based on received trades. A trade will have the *StatsIndicators* set for the statistics it affects. When a trade cancel occurs the affected *MDEntryType* will also be sent with its new value. E.g. if a cancelled trade would affect the high price a new high price is sent directly after the trade cancellation. This way the receiver do not have to calculate the statistics based on cancelled trades, only new trades.

High Price ('7') *MDEntryPx* contains the price. Updated when *StatsIndicators* contains *StatsType* "High/Low Price".

Low Price ('8') *MDEntryPx* contains the price. Updated when *StatsIndicators* contains *StatsType* "High/Low Price".

First Price ('x') *MDEntryPx* contains the price. Updated when *StatsIndicators* contains *StatsType* "Exchange Last". The first price is updated

according to the trade time (*TransBkdTime* if present, otherwise *TransactTime*) of trades (which need not be delivered in this order in case of manually reported trades). *TransactTime* contains the first execution time.

Last Price ('y') *MDEntryPx* contains the price. Updated when *StatsIndicators* contains *StatsType* "Exchange Last". The last price is updated according to the trade time (*TransBkdTime* if present, otherwise *TransactTime*) of trades (which need not be delivered in this order in case of manually reported trades). *TransactTime* contains the last execution time.

VWAP Turnover/Volume ('w') *MDEntryPx* and *MDEntrySize* contains the turnover and trade volume. The actual VWAP is calculated as the turnover divided by the volume. Updated when *StatsIndicators* contains *StatsType* "Average Price".

Trade Volume ('B') *MDEntrySize* contains the trade volume. Updated when *StatsIndicators* contains *StatsType* "Turnover".

Late Trade Volume ('u') The trade volume of late reported trades, e.g. from previous day or session. *MDEntrySize* contains the trade volume. Updated when *StatsIndicators* contains *StatsType* "Late Turnover". Note: This value can be negative, e.g. if a trade from previous day or session is cancelled.

Turnover ('z') *MDEntryPx* contains the turnover. Updated when *StatsIndicators* contains *StatsType* "Turnover".

Late Turnover ('v') The turnover of late reported trades, e.g. from previous day or session. *MDEntryPx* contains the turnover. Updated when *StatsIndicators* contains *StatsType* "Late Turnover". Note: This value can be negative, e.g. if a trade from previous day or session is cancelled.

For any auction, *opening auction, closing auction* or *circuit breaker auction*, the equilibrium price, available bid and ask volume are continuously disseminated during and upon entry of the auction for each orderbook. The equilibrium price with available buy and sell volume are updated every time there is a change in an orderbook but no more than once per second per orderbook. In the case where an orderbook is not crossed, the fields equilibrium price and volume are absent (null).

Both MDEntries *Equilibrium Buy* and *Equilibrium Sell* are sent synchronously in pairs for each orderbook.

Equilibrium Buy ('b') If the orderbook is crossed *MDEntryPx* contains the equilibrium price and *MDEntrySize* contains available buy volume at equilibrium price, otherwise *MDEntryPx* and *MDEntrySize* are absent (null).

Equilibrium Sell ('s') If the orderbook is crossed *MDEntryPx* contains the equilibrium price and *MDEntrySize* contains available sell volume at equilibrium price, otherwise *MDEntryPx* and *MDEntrySize* are absent (null).

4.6.1 MDEntry Component Block

This component block is used to define a market data entry, e.g. an order, trade or closing price.

Table 4.18: MDEntry.

Tag	*Field Name*	*Type*	*Req*	*Description*
269	MDEntryType	uInt32	Y	ASCII char enumeration. '0'=Bid '1'=Offer '2'=Trade '4'=Opening Price '5'=Closing Price '7'=Trading Session High Price '8'=Trading Session Low Price 'B'=Trade Volume 'u'=Late Trade Volume 'v'=Late Turnover 'w'=VWAP Turnover/Volume 'x'=First Price 'y'=Last Price 'z'=Turnover 'b'=Equilibrium Buy 's'=Equilibrium Sell 'r'=Accrued Interest Rate (100 = 100
336	TradingSessionID	string	N	Identifier for trading session. 1=Day (regular session) 6=After-hours (non-regular session)
20016	MDStatScope	uInt32	N	Defines the scope of the statistics in periods of time. Custom field. 1=Session 2=Day 3=Official Day

Table 4.18: MDEntry.

Tag	*Field Name*	*Type*	*Req*	*Description*
270	MDEntryPx	decimal	C	Entry price. Conditionally required in MDEntryIncr when MDUpdateAction is '0' (New) and MDEntryType is '2' (Trade), 'v' (Late Turnover), 'w' (VWAP Turnover/Volume) or 'z' (Turnover).
271	MDEntrySize	decimal	C	Conditionally required in MDEntryIncr when MDUpdateAction is '0' (New) and MDEntryType is '0' (Bid), '1' (Offer), '2' (Trade), 'B' (Trade Volume) or 'w' (VWAP Turnover/Volume). Entry quantity.
278	MDEntryID	string	N	Refers to previous MDEntryID when MDUpdateAction=Change or Delete.
290	MDEntryPositionNo	uInt32	N	Display position of a bid or offer within a price level, numbered from most competitive to least competitive, per market side, beginning with 1. This value is only set when MDUpdateAction is New or Change and only if the value has changed.
288	MDEntryBuyer	string	N	Marketplace assigned member code. Reveals the buyer when MDEntryType is Bid or Trade and counterparties are not hidden in the security.

Table 4.18: MDEntry.

Tag	*Field Name*	*Type*	*Req*	*Description*
289	MDEntrySeller	string	N	Marketplace assigned member code. Reveals the seller when MDEntryType is Offer or Trade and counterparties are not hidden in the security.
574	MatchType	uInt32	N	Match type for trades. ASCII char enumeration. '1'=One-Party Trade Report (privately negotiated trade) '2'=Two-Party Trade Report (privately negotiated trade) '4'=Auto-match '7'=Call Auction 'x'=Manually Matched Trade Report
828	TrdType	uInt32	N	Trade type for trades. 0=Regular Trade 52=Exchange Granted Trade

Table 4.18: MDEntry.

Tag	*Field Name*	*Type*	*Req*	*Description*
277	TradeCondition	string	N	Trade conditions set by exchange. Multiple char value (delimited with space). 0=Cancel (only used in snapshot) I=Sold Last (late reporting) AV=Outside Spread X0=Outside Spread Unknown XB=Knockout buyback Trade XS=Buyback Trade XD=Distribution Trade XAO=Opening auction Trade XAC=Closing auction Trade XAD=Circuit breaker dynamic auction Trade XAS=Circuit breaker static auction Trade XAP=Order protection auction Trade XAR=Missing reference price auction trade 6=Benchmark trade. MiFID II regulatory field
1839	TrdPriceCondition	uInt32	N	Applies only to manual trades. MiFID II regulatory field. 13=Special dividend Trade. 15=Non-price forming Trade. 16=Trade not contributing to the price discovery process
2667	AlgorithmicTrd-Indicator	uInt32	N	MiFID II regulatory field. Absence means '0'. 0=Non-algorithmic trade 1=Algorithmic trade

Table 4.18: MDEntry.

Tag	*Field Name*	*Type*	*Req*	*Description*
1115	OrdCategory	uInt32	N	Applies only to manual trades. MiFID II regulatory field. 3=Privately Negotiated Trade
2668	TrdRegPublications	sequence	N	Applies only to manual trades. MiFID II regulatory field.
2669	→TrdRegPublication-Type	uInt32	N	0=Pre-trade transparency waiver
2670	→TrdRegPublReason	uInt32	N	0=No preceding order in book as transaction price set within average spread of a liquid instrument. ESMA RTS "NLIQ". 1=No preceding order in book as transaction price depends on system-set reference price for an illiquid instrument. ESMA RTS "OILQ". 2=No preceding order in book as transaction price is for transaction subject to conditions other than current market price. ESMA RTS "PRIC".
1093	LotType	uInt32	N	Defines the lot type assigned to the order. ASCII char enumeration. '1'=Odd Lot '2'=Round Lot
60	TransactTime	uInt64	N	UTC timestamp when the trade was executed or when the order was created, updated or cancelled. For official statistics this denotes the time of calculation. Field added (partially).

Table 4.18: MDEntry.

Tag	*Field Name*	*Type*	*Req*	*Description*
483	TransBkdTime	uInt64	N	UTC timestamp the trade was booked, if other than Transact-Time. Used for manual trade reports. Field added (partially).
5797	AggressorSide	uInt32	N	Indicates which side is aggressor of the trade. If there is no value present, then there is no aggressor. ASCII char enumeration. Custom field. '1'=Buy '2'=Sell
20033	MarketMakerQuote	uInt32	N	Field added.. Indicates that this MDEntry originates from a Market Maker quote. Only applicable if MDEntryType = '0', '1' or '5'. ASCII char enumeration (boolean). Absence means 'N'. 'N'=Not Market Maker Quote 'Y'=Market Maker Quote

4.6.2 Market Data Request (V)

Market data (orders, trades, etc.) can be requested with the *Market Data Request* message. The reply is one or more *Market Data Snapshot Full Refresh* messages. Requested market data types (for example bid and offers or trades) must be specified through specifying one or more Market Data Entry Types. Only trades for the last 72 hours are available. Note that a reply with 0 repeating market data entries may be sent as a reply. The last *Market Data Snapshot Full Refresh* message will always be indicated with the *LastRptRequested* field set to 'Y'. In the unlikely event that there are no securities defined a dummy *Market Data Snapshot Full Refresh* message with *SecurityID* absent (null) and *LastRptRequested* field set to 'Y' will be sent as a response.
Parallel requests with equal *MDReqID* will be rejected, the requester should either use a unique *MDReqId* for each request or perform the requests sequentially.

In the event of a malformed request, the response will be a *Market Data Request Reject* message.

MarketDataRequest:

- is replied to with a *MarketDataSnapshotFullRefresh* message, with MDReqID set to (copied from) the value in the request message
- can be rejected with a *MarketDataRequestReject* message, with MDReqRejReason set to the reject reason and MDReqID set to (copied from) the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to V
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the MarketDataRequest message

Table 4.19: MarketDataRequest (V).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
262	MDReqID	string	Y	Unique identifier for Market Data Request.
263	SubscriptionRequestType	uInt32	Y	ASCII char enumeration. '0'=Snapshot
264	MarketDepth	uInt32	Y	Valid values: 0=Full book
267	MDEntryTypes	sequence	Y	Requested entry types. Empty list means all entry types.

Table 4.19: MarketDataRequest (V).

Tag	*Field Name*	*Type*	*Req*	*Description*
269	→MDEntryType	uInt32	Y	ASCII char enumeration. '0'=Bid '1'=Offer '2'=Trade '4'=Opening Price '5'=Closing Price '7'=Trading Session High Price '8'=Trading Session Low Price 'B'=Trade Volume 'u'=Late Trade Volume 'v'=Late Turnover 'w'=VWAP Turnover/Volume 'x'=First Price 'y'=Last Price 'z'=Turnover 'r'=Accrued Interest Rate (100 = 100
580	Dates	sequence	N	Range of dates for requested trades. *Since* (NoDates=1) or *Between* (NoDates=2) dates, inclusive. Sequence added.
60	→TransactTime	uInt64	Y	UTC timestamp the trade was executed.

4.6.3 Market Data Snapshot Full Refresh (W)

Response to a *Market Data Request*.

MarketDataSnapshotFullRefresh is sent:

- in reply to a *MarketDataRequest* message, with MDReqID set to (copied from) the value in the request message

Table 4.20: MarketDataSnapshotFullRefresh (W).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			

Table 4.20: MarketDataSnapshotFullRefresh (W).

Tag	*Field Name*	*Type*	*Req*	*Description*
262	MDReqID	string	C	Conditionally required when this message is a response to a request.
	component block <SecurityRef>			
268	MDEntries	sequence	Y	
	→ **component block <MDEntry>**			
912	LastRptRequested	uInt32	N	Field added.. Indicates that this is the last report which will be returned as a result of the request. ASCII char enumeration (boolean). 'N'=Not Last Message 'Y'=Last Message

4.6.4 Market Data Incremental Refresh (X)

Incremental (unsolicited) update of market data.

MarketDataIncrementalRefresh is sent:

- unsolicited, when a public change occurs in the market, for example order updates, new trades, etc.

Table 4.21: MarketDataIncrementalRefresh (X).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
268	MDEntries	sequence	Y	
279	→MDUpdateAction	uInt32	Y	ASCII char enumeration. '0'=New '1'=Change '2'=Delete
	→ **component block <SecurityRef>**			
	→ **component block <MDEntry>**			
1175	→StatsIndicators	sequence	N	
1176	→→StatsType	uInt32	Y	Type of statistics. 1=Exchange Last 2=High / Low Price 3=Average Price (VWAP, TWAP ...) 4=Turnover 100=Late Turnover

4.6.5 Market Data Request Reject (Y)

Reject of a *Market Data Request* in case of a malformed request.

MarketDataRequestReject is sent:

- to reject a *MarketDataRequest* message, with MDReqRejReason set to the reject reason and MDReqID set to (copied from) the value in the request message

Table 4.22: MarketDataRequestReject (Y).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
262	MDReqID	string	Y	Refers to the request.
281	MDReqRejReason	uInt32	N	ASCII char enumeration. '1'=Duplicate MDReqID '2'=Insufficient Bandwidth '3'=Insufficient Permissions '4'=Unsupported SubscriptionRequestType '5'=Unsupported MarketDepth '6'=Unsupported MDUpdateType '8'=Unsupported MDEntryType 'A'=Unsupported Scope 'x'=Invalid
58	Text	string	N	Error message.

4.7 Corporate Action Messages

4.7.1 Corp Action Component Block

This component block defines a corporate action, such as a split. The corporate action message defines a corporate action and it's parameters while the flag in the security status is meerly an indicator for the trader to be observant of events that will or recently has occured. Notice that a corporate action that has been executed may never be deleted and only the description may be modified.

Table 4.23: CorpAction.

Tag	*Field Name*	*Type*	*Req*	*Description*
20004	CorpActionType	uInt32	N	The type of corporate action. Custom field. 0=Cash dividend 1=Split 2=Reverse-split 3=Rights issue 99=Other
20005	CorpActionID	string	N	Unique identifier for this corporate action event. Custom field.
20008	CorpActionDescr	string	N	Textual description of the corporate action. Custom field.
20010	CorpActionStatus	uInt32	N	Custom field. 0=Not executed 1=Executed
20017	ExTime	uInt64	N	UTC timestamp when this corporate action takes effect. Custom field.
60	TransactTime	uInt64	N	UTC timestamp this corporate action was created or updated.
20006	AdjustmentFactor-Numerator	uInt32	N	The adjustmentfactor of a corporate action is the numerator divided by the denominator and is used when adjusting historical values for the corporate action. Prices should be multiplied with the factor while quantities should be divided by the factor. Custom field.

Table 4.23: CorpAction.

Tag	*Field Name*	*Type*	*Req*	*Description*
20022	AdjustmentFactor-Denominator	uInt32	N	The adjustmentfactor of a corporate action is the numerator divided by the denominator and is used when adjusting historical values for the corporate action. Prices should be multiplied with the factor while quantities should be divided by the factor. Custom field.
20007	Dividend	decimal	N	Dividend, 3 decimal precision. Custom field.

4.7.2 Corporate Action Report (U1)

The *Corporate Action Report* is used for unsolicited updates of corporate actions and as a response to a *Corporate Action Request*. The field *CorpUpdateAction* is absent (null) in a snapshot response.

CorporateActionReport is sent:

- unsolicited, when a change occurs
- in reply to a *CorporateActionRequest* message, with CorpActionResult set to 0 (Succeeded) and CorpActionReqID set to (copied from) the value in the request message
- to reject a *CorporateActionRequest* message, with CorpActionResult set to 1 (InvalidRequest) and CorpActionReqID set to (copied from) the value in the request message

Table 4.24: CorporateActionReport (U1).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
	component block <SecurityRef>			
20009	CorpActionReqID	string	N	Unique request identifier. Custom field.

Table 4.24: CorporateActionReport (U1).

Tag	*Field Name*	*Type*	*Req*	*Description*
20012	CorpActionResult	uInt32	N	Result returned to a Corporate Action Request message. Custom field. 0=Succeeded (default) 1=Invalid or unsupported request
912	LastRptRequested	uInt32	N	Indicates that this is the last report which will be returned as a result of the request. ASCII char enumeration (boolean). 'N'=Not Last Message 'Y'=Last Message
20011	CorpUpdateAction	uInt32	N	The update action of an incremental update. Absent in a snapshot response. ASCII char enumeration. Custom field. 'A'=Add 'D'=Delete 'M'=Modify
	component block <CorpAction>			

4.7.3 Corporate Action Request (U2)

All corporate actions can be requested with the *Corporate Action Request* message. The reply is one or more *Corporate Action Report* messages. The last *Corporate Action Report* message will always be indicated with the *LastRptRequested* field set to 'Y'. In the event that there are no corporate actions a dummy *Corporate Action Report* message with *SecurityID* absent (null) and the *LastRptRequested* field set to 'Y' will be sent as a response. All planned and already executed Corporate Actions will be sent.

In the event of a malformed request, the response will be a *Corporate Action Report* message with the *CorpActionResult* field set to 1 (Invalid or unsupported request).

CorporateActionRequest:

- is replied to with a *CorporateActionReport* message, with CorpActionResult set to 0 (Succeeded) and CorpActionReqID set to (copied from) the value in the request message

- can be rejected with a *CorporateActionReport* message, with CorpActionResult set to 1 (InvalidRequest) and CorpActionReqID set to (copied from) the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to U2
- can be rejected with a *Reject* message, with SessionRejectReason set to the reject reason and RefSeqNum set to the sequence number of the CorporateActionRequest message

Table 4.25: CorporateActionRequest (U2).

Tag	*Field Name*	*Type*	*Req*	*Description*
	component block <StandardHeader>			
20009	CorpActionReqID	string	Y	Unique request identifier. Custom field.

Chapter 5

FAST Encoding and Templates

The FIX messages described in this specification are encoded with FAST 1.1[1] meaning that the traditional ASCII encoding ("Tag=Value") is not supported. FAST SCP (Session Control Protocol) 1.1 level 2 is used as a thin layer on top of TCP which is used as the transport protocol. The FAST SCP 1.1 level 2 provides messages like *Hello*, *Alert* and *Reset* for logon, notification and FAST specific functionality such as dictionary reset.

A FAST stream can be sent as a sequence of messages or *blocks* where each block consists of a sequence of messages, in addition a *block size* is preceding each block. NGM uses blocks with one message per block. The block size value specifies the size in bytes of the following message, not including the size of the actual block size field. According to FAST 1.1[2], the block size should be an unsigned interger that may be overlong, NGM has chosen to encode the block size as a 4 byte overlong unsigned integer.

The FIXT (FIX Transport) session messages (see section 2.5) are used for maintaining FIX sessions, which are typically long lived sessions spanning several FAST SCP/TCP connections.

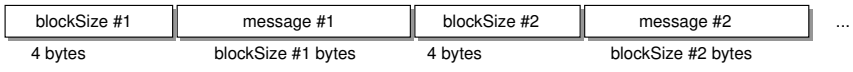

Figure 5.1: FAST block size.

5.1 Data Types

This section describes how FIX data types are encoded in FAST. Note that the *type* column in the message tables contains the FAST type that is used.

[1] See FAST Specification 1.x.1, http://www.fixtradingcommunity.org/pg/structure/tech-specs/fast-protocol

[2] See FAST Specification 1.x.1 chapter 10

Note that 32 and 64-bit unsigned integers only support 31 and 63 bits respectively.

5.1.1 Strings

All *non-encoded* strings are treated as ASCII (7-bit). Non-ASCII strings can be sent in the encoded fields (e.g. EncodedIssuer). The only supported encoding for those fields is UTF-8 (unicode string in FAST). The encoded representation is only present when the encoded value differs from the non-encoded value. Whenever the encoded value is present the non-encoded value is also present.

XML data fields (e.g. SecurityXML) are UTF-8 encoded (unicode string in FAST).

5.1.2 Identifiers

Any identifiers that are ASCII strings are accepted. Identifiers generated by NGM are restricted to contain `A-Z`, `0-9` and the characters `+-:.,?` and the maximum length is 16.

5.1.3 Enumerations

In FIX several types are used for enumerations: integer, ASCII char and ASCII string. In most cases when a string is used for enumerations only one character is used, which means it can easily be reduced to the char case.

In FAST both integer enumerations and ASCII char enumerations are encoded as unsigned 32-bit integers, while in the ASCII char enumeration case the value will always be an ASCII character code. In the documentation these enum types will be differentiated by single quotes around the ASCII char enums, e.g. '1' means 49.

5.1.4 Timestamps and Dates

UTC timestamps fields are encoded as microseconds since January 1, 1970 UTC, as a 64-bit unsigned integer. *Leap seconds are not added*, which is the same way as POSIX (and Java as it seems) does it. This means that all days have exactly $24*3600*1000000$ microseconds. For example the timestamp difference between 31 dec 2008 23:59:00 and 1 jan 2009 00:01:00 is reported as 120 seconds and not the correct UTC difference which is 121 seconds since a leap second should be added 31 dec 2008 23:59:60.

Other dates and timestamps than UTC are encoded as ASCII string as specified in FIX.

5.2 Templates

The FAST templates specifies how messages are encoded. Static FAST templates are used and any changes to the templates are considered a protocol change.

FAST templates need to be mapped to FIX messages. The following mapping rules are used.

- Message level: FIX message name as appearing in the FIX repository (e.g. "NewOrderSingle") = FAST application type (typeRef).
- Field level: FIX field tag = FAST field aux id.
- Type conversion: No type conversion is made. E.g. a FIX field of string type requires that the corresponding FAST field is also of string type.
- Missing fields in FAST: If a FIX field is missing in the FAST template, the field is assumed to be absent. This is only valid for optional FIX fields.
- Sequence fields: Sequence fields in FAST are mapped to the corresponding *NoXXX* field in FIX, e.g. for *NoSides* (552) the FAST sequence aux id should be 552.
- Group fields in FAST: FAST group fields are flattened before mapping to FIX.
- Dynamic template ref in FAST: Not supported.

Because of this mapping, the FIX field *MsgType* is not really required for message type identification.

Appendix A

MiFID II Regulatory fields

A.1 Post trade transparency

MiFID II regulatory post-trade information mapping against FIX fields.

- *BENCH*
  - Private service: SecondaryTrdType(855) = 64 (Benchmark trade)
  - Public service: TradeCondition(277) = 6 (Benchmark trade)
- *NPFT*
  - TrdPriceCondition(1839) = 15 (Non price forming trade)
- *TNCP*
  - TrdPriceCondition(1839) = 16 (Trade not contributing to the price discovery process)
- *SDIV*
  - TrdPriceCondition(1839) = 13 (Special dividend trade)
- *ALGO*
  - AlgorithmicTrdIndicator(2667) = 1 (Algorithmic trade)
- *NLIQ*
  - TrdRegPublicationType(2669) = 0 (Pre-trade transparency waiver)
  - TrdRegPublicationReason(2670) = 0 (No preceding order in book as transaction price set within average spread of a liquid instrument)
- *OILQ*
  - TrdRegPublicationType(2669) = 0 (Pre-trade transparency waiver)
  - TrdRegPublicationReason(2670) = 1 (No preceding order in book as transaction price depends on system-set reference price for an illiquid Instrument)

- *PRIC*
  - TrdRegPublicationType(2669) = 0 (Pre-trade transparency waiver)
  - TrdRegPublicationReason(2670) = 2 (No preceding order in book as transaction price is subject to conditions other than current market price)

A.2 Order Record Keeping

A.2.1 Description of the different party roles

For EU markets it is mandatory to provide party information on orders and quotes and the information in this chapter applies. If not sure, consult the Market Model or the market place for information on whether it is required to supply party information.

- Only identifiers in the form of short codes are allowed to be sent over the NGM FIX Protocol.
- PartyID values 0-10 are reserved and must not be used to identify any party.
- The short code together with the PartyRoleQualifier is the unique identifier for a mapping.
- Information on the mapping between a short code + role (PartyRoleQualifier) and the actual identifier (National ID, LEI and Algorithm ID) must:

  - never change over time

  - be provided separately, outside of the NGM FIX Protocol,

  - have been supplied before to the first usage of the short code in the protocol, or latest by the end of the actual calendar day that the short code is first used (see the Market Model for details).

Client Identification (PartyRole = 3) Used to identify the client of the member or participant of the trading venue.

- In case of that there is no client for an order, the PartyID should be set to *0 (=NONE)* for PartyRole = 3.
- In case of aggregated orders, the PartyID should be set to 1 *(=AGGR)* for PartyRole = 3.
- In case of pending allocations, the PartyID should be set to 2 *(=PNAL)* for PartyRole = 3.

Executing Trader (PartyRole = 12) Used to identify the person or algorithm within the member or participant of the trading venue who is responsible for the execution of the transaction resulting from the order or the quote. Executing Trader is required to be specified on all orders and quotes.

- In case of the time and venue of the order is instructed by the client of the member or participant of the trading venue the PartyID should be set to 3 *(=CLIENT)* for PartyRole = 12.

Investment Decision Maker (PartyRole = 122) Used to identify the person or the algorithm within the member or participant of the trading venue who is responsible for the investment decision.

A.2.2 Orders

- Party information is required on the first submission of an order (New Order Single)
- Party information is not possible to change after the first submission.
- Party information is acknowledged in ExecutionReports.
- If a PartyRole is populated in an order, it is required that the accompanying fields PartySourceID, PartyID and PartyRoleQualifier are also populated.
- Client identification is mandatory for orders.
- Executing Trader (PartyRole = 12) is mandatory for orders.
- Investment Decision Maker (PartyRole = 122) shall not be set when the investment decision was not made by a person or algorithm within the member or participant of the exchange.

A.2.3 Quotes

- Party information is required on the first entry of a quote
- Party information must not be set in subsequent updates of the quote.
- If party information is supplied in updates of a quote, then the update is rejected.
- Party information is only acknowledged in the first QuoteStatusReport.
- If a PartyRole is populated in a quote, it is required that the accompanying fields PartySourceID, PartyID and PartyRoleQualifier are also populated.
- Executing Trader (PartyRole = 12) is mandatory for quotes.
- If the Investment Decision Maker (PartyRole = 122) is not set, the PartyRoleQualifier, PartyIDSource and PartyID values for the Executing Trader will be implicitly used for the identification of the Investment Decision Maker.

Appendix B

Manually Matched Orderbooks

B.1 Overview

Orders in a manually matched orderbook are not matched automatically. This is instead handled by a designated broker trader group by using the *Trade Capture Report* message with the *Manual Match Report* model.

B.2 Manual Match Report

In the manual match report model *no* Trade Capture Report Ack is sent in response to a successful request. The confirmed trade is sent directly instead. The fields are used in the following way in this model.

Designated broker submit to marketplace.

Trade Capture Report
TradeReportTransType = New (0)
TradeReportType = Submit (0)
TradeHandlingInstr = Two-Party Report ('1')
MatchType = Manually Matched Trade Report ('x')
LastPx = <trade price>
LastQty = <trade volume>
Sides =
- Side = Buy ('1')
  OrderId = <reference>
- Side = Sell ('2')
  OrderId = <reference>

TradeReportID=<new>

Marketplace confirm trade to buyer and seller.	**Trade Capture Report** TradeReportTransType = New (0) TradeReportType = Submit (0) TradeHandlingInstr = Trade Confirm ('0') MatchType = Manually Matched Trade Report ('x') TradeReportID=<new> TradeID=<reference> MatchStatus = Affirmed ('0')
Marketplace confirm trade to designated broker.	**Trade Capture Report** TradeReportTransType = New (0) TradeReportType = Submit (0) TradeHandlingInstr = Trade Confirm ('0') MatchType = Manually Matched Trade Report ('x') TradeReportRefID=<designated broker's> TradeReportID=<new> TradeID=<reference> MatchStatus = Affirmed ('0')
Reject from marketplace in response a malformed Trade Capture Report.	**Trade Capture Report Ack** TradeReportTransType = <same> TradeReportType = <same> TradeHandlingInstr = Two-Party Report ('1') TradeReportRefID=<same> TradeReportID=<same> TradeReportRejectReason=<specified>