Elasticia FIX Protocol

Version 3.0.0 - 2026-05-26

Contents

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 (FIX Latest) for application level messages.
- FIX session protocol FIXT 1.1 for maintaining FIX sessions.
- FIX Classic (tag-value) is supported for message encoding.
- FAST 1.1 (FIX Adapted for STreaming) is supported for message encoding. In this case 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; *Order Entry* for order management, order status, trade reporting and similar tasks, and a *Market Data* 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.

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 the FIX protocol specification is unclear or something must be bilaterally agreed it is also described in this document.

Section 1 (this section) gives an overview of the NGM FIX protocol.

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

Section 3 explains the order entry service which is used for orders, quotes and trades.

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

Section 5 explains how regulatory fields are used.

2. General Service Information

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

2.1. Data Types

Throughout this document, the FIX data types are used for documentation in message tables, with the following exceptions and clarifications:

- **uint32** and **uint64**: corresponds to FIX type int and FAST types uInt32 and uInt64.
- **decimal**: corresponds to FIX type float and FAST type decimal.
- **String**: Any 7-bit ASCII except the <SOH> delimiter (0x01). Corresponds to FIX type String and FAST type String with charset "ascii" (7-bit).
- **UnicodeString**: Unicode string that corresponds to FIX types data and XMLData (UTF-8), and FAST type String with charset "unicode".
- **char**: mapped to FAST uInt32 containing the ASCII value of the char.
- **UTCTimestampMicros**: corresponds to FIX UTCTimestamp (with micro second resolution) and FAST uInt64 encoded as microseconds since January 1, 1970 UTC, without leap seconds (POSIX compliant).
- **Length**: A uInt32 value that specifies the number of bytes in the corresponding data field.

In FIX several types are used for enumerations: integer, char and String. In the documentation these enum types will be dif-

ferentiated by single quotes around char enums, e.g. '1' means 49, and double quotes around String enums.

2.1.1. Identifiers and Maximum String Lengths

Identifiers generated by the exchange, such as `OrderID` and `ExecID`, only contain characters `A-Z`, `0-9` and `+-:.,?` with the maximum length 16.

The member code is restricted to uppercase alphanumeric characters (A-Z,0-9), with maximum length 16, at a technical level. Market model rules may further restrict this.

The following client-assigned fields are restricted to 7-bit ASCII printable characters (0x20 - 0x7f), with maximum lengths as defined below:

- `ClOrdID` 14 bytes.
- `QuoteMsgID` 14 bytes.
- `TradeReportID` 14 bytes.
- `Account` 255 bytes.
- `PartySubID` person in one-party-for-pass-thru trades, 255 bytes.

> **Note**
>
> The fields `ClOrdID`, `QuoteMsgID` and `TradeReportID` are further restricted in uniqueness in Section 3.3, “Idempotent Operations”.

2.2. FAST Encoding

FAST 1.1 message encoding is provided. 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, the block size should be an unsigned integer that may be overlong, NGM has chosen to encode the block size as a 4 byte overlong unsigned integer.

2.2.1. FAST 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 auxiliary identifier.
- 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.
- Extra fields in FAST: If the FAST template contains a field that cannot be mapped to a FIX field, it is parsed and ignored.
- Sequence fields: Sequence fields in FAST are mapped to the corresponding *NoXXX* field in FIX, e.g. for *NoSides* (552) the FAST sequence auxiliary identifier should be 552.
- Group fields in FAST: FAST group fields are flattened before mapping to FIX.
- Dynamic template ref in FAST: Not supported/used.

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

2.3. 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 *NextExpectedMsgSeqNum* field. Note that the *ResendRequest* message is not supported. See Section 2.7.1, “Logon (A)” 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 intraday. 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.

2.4. 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.

For each data class, the following filter rules exist (based on roles):

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.

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 except snapshot replies, rejects 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.

2.5. Throttling Limits

Each FIX session has throttling limits on:

- Inbound rate
- Outstanding requests

The inbound rate throttle, 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). When the throughput exceeds the limit, a *Business Message Reject* message is sent, and the referred message and any additional messages are *delayed* until the next second.

The outstanding request throttle, limits the number of outstanding requests that can be sent to the exchange, without receiving a response on the previous requests. The outstanding request counter is calculated in the FIX gateway, and incremented on requests and decremented on responses. When the number of outstanding requests exceeds the limit, a *Business Message Reject* message is sent (max once a second), and the referred message and any additional messages are *delayed* until any previous request has got a response.

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 throttle limits that are used for your FIX session is only available *offline* (outside the protocol), i.e. contact the exchange for more information.

2.6. Component Blocks

2.6.1. Standard Header

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

The *CompID* fields denotes the member or trader group on one side, and the marketplace or market data channel on the other side. The *Sender-* and *TargetCompID* pair identifies a FIX session.

For inbound messages (to the marketplace):

- *SenderCompID* denotes the member or trader group.
- *TargetCompID* denotes the marketplace (or market data channel).

For outbound messages (from the marketplace):

- *SenderCompID* denotes the marketplace (or market data channel).
- *TargetCompID* denotes the member or trader group.

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

- *SenderCompID* denotes the member or trader group of the service connection.
- *TargetCompID* denotes the marketplace (or market data channel).
- *OnBehalfOfCompID* denotes the member or trader group of the origin firm.

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

- *SenderCompID* denotes the marketplace (or market data channel).
- *TargetCompID* denotes the member or trader group of the service connection.
- *DeliverToCompID* denotes the member or trader group of the destination firm.

Tag	Field Name	Type	Req
34	MsgSeqNum	uint64	Y
	Message sequence number.		
49	SenderCompID	String	Y
	Identifies sender firm (and trader group).		
56	TargetCompID	String	Y
	Identifies target firm (and trader group).		
115	OnBehalfOfCompID	String	N
	Identifies sending firm, used when sending messages via a third party.		
128	DeliverToCompID	String	N
	Identifies target firm, used when sending messages via a third party.		
52	SendingTime	UTCTimestampNanos	Y
	Time of message transmission.		

2.6.2. 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.

Tag	Field Name	Type	Req
48	SecurityID	String	N
	Security identifier of type specified in SecurityIDSource.		
22	SecurityIDSource	char	N
	Identifies the class of SecurityID. Only Marketplace-assigned identifier is allowed in this context.		

Tag	Field Name	Type	Req
	'M'=Marketplace-assigned identifier *'4'=ISIN* *'8'=Exchange Symbol* *'D'=Valoren*		

2.7. Session Messages

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

FIX session sequence numbers (*MsgSeqNum*) starts at 1 and are normally never reset by the exchange, not even at midnight. Instead, they are incremented forever. 24/7 connectivity is supported, but *MsgSeqNum* cannot be 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) at logon if desired. The *MsgSeqNum* is represented as a 64-bit integer.

The *NextExpectedMsgSeqNum* 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.

SendingTime is validated so that it is inside of the SendingTimeThreshold. Ngm uses a SendingTimeThreshold of 1 hour. On failure the connection will be closed.

2.7.1. 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 configured for the FIX session.

TargetCompID As configured for the FIX session.

Username Specifies the user to logon.

The *Logon* message is a part of the message recovery mechanism. The *NextExpectedMsgSeqNum* 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 2.3, “Recovery”.

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

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

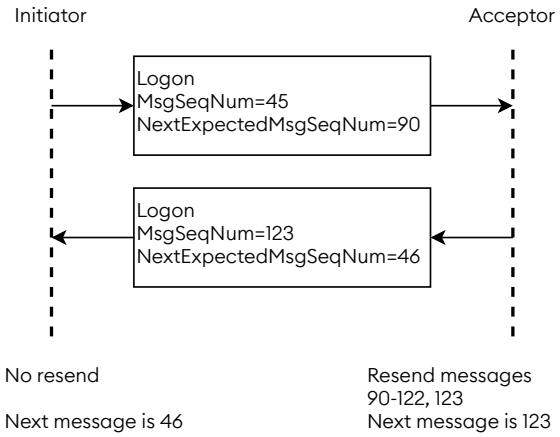
MsgSeqNum is too low the session state is broken. This indicates some kind of error (e.g. software error, human error).

Incorrect reset the 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.

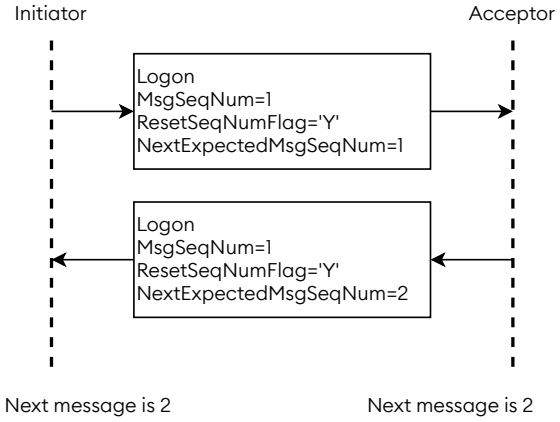
The figure below shows an example logon scenario. Any messages that need to be resent are sent directly after the logon messages has been exchanged. The *Logon* message with *MsgSeqNum=123* is resent as a gap-fill directly after the messages 90-122 have been resent.

Figure 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 below). 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. 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 the reject reason

- 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

Tag	Field Name	Type	Req
	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: "10" = FIXLatest.		
141	ResetSeqNumFlag	char	N
	Indicates both sides of a FIX session should reset sequence numbers. Absence means 'N'. *'N'=No* *'Y'=Yes, reset sequence numbers*		
789	NextExpectedMsgSeqNum	uint64	Y
	Message sequence number gap detection.		
553	Username	String	N
554	Password	String	N

2.7.2. 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 the reject reason

Tag	Field Name	Type	Req
	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

2.7.3. Test Request (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 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.

Tag	Field Name	Type	Req
	component block <StandardHeader>		
112	TestReqID	String	Y

2.7.4. 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 the value in the request message

Tag	Field Name	Type	Req
	component block <StandardHeader>		
112	TestReqID	String	N

2.7.5. SequenceReset (4)

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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
123	GapFillFlag	char	N
	'Y'=Gap Fill Message, Msg Seq Num Field Valid		
36	NewSeqNo	uint64	Y
	Next sequence number.		

2.7.6. 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

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

2.8. General Application Level Messages

2.8.1. 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

Tag	Field Name	Type	Req
	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* *8=Throttle limit exceeded* *18=Invalid price increment*		
58	Text	String	N
	Where possible, message to explain reason for rejection		

3. Order Entry Service

The order entry 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 order entry service is divided into three levels; organization, trader group and user. Within the organi-

zation 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.

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 other's operations.

Each user has a separate FIX session to the private service. A snapshot request will only affect the session 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 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. Idempotent Operations

Order, quote and trade reporting operations are idempotent using the client assigned identifiers `ClOrdID`, `QuoteMsgID` and `TradeReportID`.

The identifiers must be unique within:

- the trader group,
- the security,
- the operation category (order, quote or trade reporting), and
- the last 24 rolling hours.

If e.g. an order operation is using a `ClOrdID` value that has already been seen before, within the configured time window, the operation will be rejected with a specific error code. This guarantees at-most-once execution of these operations.

The idempotent operations include:

- New Order Single
- Order Cancel Replace Request
- Order Cancel Request
- Quote
- Quote Cancel
- Trade Capture Report

3.4. Full Snapshot Recovery

On the order entry service snapshots can be requested for the following:

Orders See the *Order Mass Status Request* message in Section 3.9.7, "Order Mass Status Request (AF)".

Quotes See the *Quote Status Request* message in Section 3.10.6, "Quote Status Request (a)". 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 cancellations.

Trades See the *Trade Capture Report Request* message in Section 3.11.6, "Trade Capture Report Request (AD)".

3.5. 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 field *OnBehalfOfCompID* to distinguish the serviced organisations when sending messages to the NGM exchange. Outbound messages will contain information in the field *DeliverToCompID* which refers to the *OnBehalfOfCompID* field of the inbound messages.

A provider may send orders, quotes and trades on behalf of another member.

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

3.5.1. Supported messages

Inbound messages allowed for usage of *OnBehalfOfCompID*:

- NewOrderSingle
- OrderCancelReplaceRequest
- OrderCancelRequest
- Quote
- QuoteCancel
- TradeCaptureReport

Outbound messages using *DeliverToCompID*:

- ExecutionReport
- TradeCaptureReport
- OrderCancelReject
- QuoteStatusReport
- BusinessMessageReject

3.6. Message Overview

The following messages can be sent/received by the client to/from the order entry service. Depending on the role only a subset of the following messages may be sent/received.

Table 1. Message overview.

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

The following are examples of roles 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: *Order=none, Quote=all, Trade=all.*

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

3.7. Component Blocks

3.7.1. PartyID

The Parties component block is used to identify and convey information on the entities both central and peripheral to the financial transaction represented by the FIX message containing the Parties Block.

Tag	Field Name	Type	Req
448	PartyID	String	Y
447	PartyIDSource	char	Y
	'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*		
452	PartyRole	uint32	Y
	3=ClientID *12=Executing trader* *122=Investment decision maker* *17=Contra Firm* *27=Buyer/Seller*		
2376	PartyRoleQualifier	uint32	N
	22=Algorithm *23=Firm or legalEntity* *24=Natural person*		
802	NoPartySubIDs	Sequence	N
523	→PartySubID	String	Y
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)*		

3.7.2. OrderAttribute

The OrderAttributeGrp component block provides additional attributes about the order.

Tag	Field Name	Type	Req
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 transaction "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	OrderAttributeValue	String	Y
	The value associated with the attribute type specified in OrderAttributeType. *Must be "Y".*		

3.8. Parties Information

Orders, quotes and trades contains parties information. The parties information can be split up in two broad data sets:

Regulatory Information — Regulatory information about the parties behind the order, quote or trade using short codes. Only revealed to the owner, and copied from orders and quotes to trade when they are matched.

Counterparty Identification — Identifies member and/or trader group of buy and sell sides of a trade. Revealed to both sides. For manual trade reporting, an optional name of the trader may be specified.

3.8.1. Regulatory Parties Information

For EU markets it is mandatory to provide party information on orders, quotes and manually reported trades. See Section 5.2, "Order Record Keeping" for more information.

The following party roles are used for regulatory party information:

- ClientID (3)
- Executing trader (12)
- Investment decision maker (122)

The regulatory party information is specified with the following fields:

- PartyIDSource (447) - Always ShortCodeIdentifier (P)
- PartyID (448) - The short code value
- PartyRoleQualifier (2376) - The role qualifier

PartySubIDs (802) is not used in this context.

3.8.2. Counterparty Identification

In Trade Capture Reports there is a need to identify the own side and the counterparty firm. This applies to the following party roles:

- Buyer/Seller (27) - Used on own trade side, where trade side details are included.
- Contra Firm (17) - The counterparty side, where trade side details are excluded.

Trade side details include order and quote references, account and regulatory parties information.

The party firm identification values are specified with the following fields:

- PartyIDSource (447) - Always CustomCode (D)
- PartyID (448) - The member code
- PartySubID (523) - Usage depends on PartySubIDType (803):
  - System (3) - The full trader group code (defaults to member code)
  - Person (2) - *Optional:* The name (or email etc.) of the trader/desk, for routing in the one-party-for-pass-thru model.

PartyRoleQualifier (2376) is not used in this context.

3.9. 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.

3.9.1. Order Component Block

This component block is used to define an order.

Tag	Field Name	Type	Req
54	Side	char	Y
	'1'=buy *'2'=sell*		
40	OrdType	char	N
	'1'=market *'2'=limit*		
44	Price	decimal	N
	Required for limit orders.		
38	OrderQty	decimal	N
1138	DisplayQty	decimal	N
	Displayed quantity on iceberg/reserve order.		
1083	DisplayWhen	char	N
	Instructs when to refresh DisplayQty. *'1'=Immediate (after each fill)* *'2'=Exhaust (when DisplayQty = 0)*		
1084	DisplayMethod	char	N
	Defines what value to use in DisplayQty. If not specified the default DisplayMethod is '1'. *'1'=Initial (use original DisplayQty)* *'2'=New (use RefreshQty)*		
1088	RefreshQty	decimal	N
59	TimeInForce	char	N
	Absence means '0'. *'0'=Session* *'1'=Good Till Cancel(GTC)* *'3'=Immediate Or Cancel (IOC)* *'4'=Fill Or Kill (FOK)* *'6'=Good Till Date (GTD)*		

Tag	Field Name	Type	Req
	'9'=AtCrossing *'B'=Good For Auction (GFA).* *An order that is valid for an auction initiated by a trading firm, see AuctionType for examples.*		
126	ExpireTime	UTCTime-stampNanos	N
60	TransactTime	UTCTime-stampNanos	N
	When this order request was created, updated or cancelled.		
1	Account	String	N
	Account information that will be echoed back.		
18	ExecInst	MultipleChar-Value	N
	Instructions for order handling (separated with spaces). *'d'=Sweep Order Book. Custom value.* *'o'=Cancel on connection loss* *'P'=Allow price adjustment on corporate action or tick size rule change. Custom value.*		
529	OrderRestrictions	MultipleChar-Value	N
	Restrictions associated with an order. *'B'=Issuer Holding* *'C'=Issue Price Stabilization*		
1803	AuctionType	uint32	N
	Conditionally required for auction orders. *100=Quote on demand auto execute or cancel. Custom value.*		
528	OrderCapacity	char	N
	Designates the capacity of the firm placing the order. Absence means 'R'. *'P'=Principal (DEAL = Deal)* *'R'=Riskless principal (MTCH = Matched)* *'A'=Agency (AOTC = 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*		
2593	NoOrderAttributes	Sequence	N
	→component block <OrderAttribute>		
453	NoPartyIDs	Sequence	N
	→component block <PartyID>		

To create a **reserve order (iceberg order)**, the field *DisplayWhen* must be present. *DisplayMethod* may be specified (otherwise its default value is used). If the *DisplayMethod* is *Initial*, either *RefreshQty* or *DisplayQty* must be given. If the *DisplayMethod* is *New*, *RefreshQty* is must be given.

3.9.2. 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 the value in the request message
- can be rejected with an *ExecutionReport* message, with ExecType set to '8' (Rejected) and ClOrdID set to 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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
11	ClOrdID	String	Y
	component block <SecurityRef>		
	component block <Order>		

3.9.3. 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 the value in the request message
- can be rejected with an *OrderCancelReject* message, with ClOrdID set to 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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
37	OrderID	String	N
41	OrigClOrdID	String	N
11	ClOrdID	String	Y
	component block <SecurityRef>		
	component block <Order>		

3.9.4. 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 the value in the request message
- can be rejected with an *OrderCancelReject* message, with ClOrdID set to 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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
37	OrderID	String	N
41	OrigClOrdID	String	N
11	ClOrdID	String	Y
	component block <SecurityRef>		
60	TransactTime	UTCTimestampNanos	Y
	When this order was cancelled.		

3.9.5. Execution Report (8)

If an order is (partially) filled upon hitting the order book only one *Execution Report* will be sent, with execution type *Trade* and order status *(Partially) Filled*.

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. However, a *Trade Capture Report* is sent for each trade, *after* the *Execution Report*. Pending order states are not used.

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 the value in the request message
- to reject a *NewOrderSingle* message, with ExecType set to '8' (Rejected) and ClOrdID set to the value in the request message
- in reply to an *OrderCancelReplaceRequest* message, with ClOrdID set to the value in the request message
- in reply to an *OrderCancelRequest* message, with ClOrdID set to the value in the request message
- in reply to an *OrderMassStatusRequest* message, with MassStatusReqID set to the value in the request message and ExecType set to 'I' (OrderStatus)

Tag	Field Name	Type	Req
	component block <StandardHeader>		
17	ExecID	String	Y
	Unique identifier of execution message, or "0" for ExecType='I' (Order Status).		
150	ExecType	char	Y
	'0'=New *'4'=Canceled* *'5'=Replaced* *'8'=Rejected* *'C'=Expired* *'D'=Restated* *'F'=Trade (partial fill or fill)* *'I'=Order Status*		
	component block <SecurityRef>		
	component block <Order>		
37	OrderID	String	Y
278	MDEntryID	String	N
	Reference to the MDEntryID of this order in the market data.		
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	char	Y
	'0'=New *'1'=Partially filled* *'2'=Filled* *'4'=Canceled* *'8'=Rejected* *'C'=Expired* *'3'=Done for day*		
636	WorkingIndicator	char	N
	Indicates if the order is currently being worked. Applicable for OrdStatus = "New" and OrdStatus = "Partially filled". Absence means 'Y'. *'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	char	N
	Defines the lot type assigned to the order. *'1'=Odd Lot* *'2'=Round Lot*		
6	AvgPx	decimal	N
	Average traded price.		
103	OrdRejReason	uint32	N
	Code to identify reason for order rejection. *1=Unknown symbol*		

Tag	Field Name	Type	Req
	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 state* *101=Buy orders not allowed in knockout buyback state* *103=Buy orders not allowed in buyback state* *104=Sell 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*		
378	ExecRestatementReason	uint32	N
	Reason for an Execution Report message sent when communicating Expired ('C') or an unsolicited cancel, or for restatement of multi-day order. *0=GT corporate action* *1=GT renewal/restatement (no corporate action)* *12=Cancel on connection loss* *99=Other* *100=Book cleared* *101=Volatility guard* *102=Cancel because of changed trading rules* *103=Cancel because of security status change (e.g. Distribution, BuyBack)* *104=Cancel because of quote validation timeout* *105=Cancel because of quote on demand not filled* *106=Expired because not filled at auction* *107=System state changed* *108=Canceled by a user other than the owner of the order or quote* *109=Canceled in immediate mode (Distribution, KnockOutBuyBack), or when IOC or FoK is canceled*		
20028	OrderPriority	uint64	N
	Indicates the priority of the order in the orderbook in comparison to other orders on the same level. Higher value means lower priority. Custom field.		
584	MassStatusReqID	String	N
	Value assigned by issuer of Mass Status Request to uniquely identify the request.		
912	LastRptRequested	char	N
	Indicates that this is the last Execution Report which will be returned as a result of the request. *'N'=Not Last Message* *'Y'=Last Message*		
58	Text	String	N
	Error message.		

3.9.6. 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 the value in the request message and CxlRejReason set to the reject reason
- to reject an *OrderCancelReplaceRequest* message, with ClOrdID set to the value in the request message and CxlRejReason set to the reject reason

Tag	Field Name	Type	Req
	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	char	Y
	If CxlRejReason=Unknown Order, value is '8'. *'0'=New* *'1'=Partially filled* *'2'=Filled* *'4'=Canceled* *'8'=Rejected* *'C'=Expired* *'3'=Done for day*		
434	CxlRejResponseTo	char	Y
	Identifies type of message this reject is in response to. *'1'=Order cancel request* *'2'=Order cancel/replace request*		
102	CxlRejReason	uint32	N
	1=Unknown order *6=Duplicate ClOrdID (11) received* *18=Invalid price increment* *99=Other* *100=Orders not allowed in knockout state* *101=Buy orders not allowed in knockout buyback state* *103=Buy orders not allowed in buyback state* *104=Sell 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*		
58	Text	String	N
	Error message.		

3.9.7. 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 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 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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
584	MassStatusReqID	String	Y
585	MassStatusReqType	uint32	Y
	7=Status for all orders		

3.10. 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 quotes are updated using *TotalBidSize* and *TotalOfferSize* to avoid the risk of over-fills, or alternatively using *BidSize* and *OfferSize*.

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, and is sent *after* the *Quote Status Report*. A completely filled quote is deleted.

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

During security status sub-state *Buyback* the exchange accepts double-sided quotes from the market maker, however the sell side of the quote is cleared. This is reflected in the *Quote Status Report* where available volume (*OfferSize*) of the sell side will be set to zero, as in a fill of that side.

Quotes always have an implicit order capacity of DEAL. For any other capacity, orders must be used.

3.10.1. Quote Grp Component Block

This component block defines a quote.

Tag	Field Name	Type	Req
132	BidPx	decimal	N
	Bid price. Either BidPx, OfferPx or both must be specified.		
133	OfferPx	decimal	N
	Offer price. Either BidPx, OfferPx or both must be specified.		
134	BidSize	decimal	N
	Specifies the open bid size. Specifies the available bid size.		
1749	TotalBidSize	decimal	N
	Specifies the total bid size.		
135	OfferSize	decimal	N
	Specifies the available ask size.		
1750	TotalOfferSize	decimal	N
	Specifies the total ask size.		
60	TransactTime	UTCTimestampNanos	N
	When 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. Absence means tradeable. *1=Tradeable.* *4=Initially tradeable (quote validation).*		
529	OrderRestrictions	MultipleCharValue	N
	Restrictions associated with an order. *'B'=Issuer Holding* *'C'=Issue Price Stabilization*		

Tag	Field Name	Type	Req
453	NoPartyIDs	Sequence	N
	→component block <PartyID>		

3.10.2. Quote (S)

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

Quote:

- is replied to with a *QuoteStatusReport* message, with QuoteMsgID set to the value in the request message
- can be rejected with a *QuoteStatusReport* message, with QuoteMsgID set to 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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
	component block <SecurityRef>		
131	QuoteReqID	String	N
117	QuoteID	String	N
	Quote identifier assigned by the exchange.		
1166	QuoteMsgID	String	Y
	Unique client-assigned identifier for the (replacement) quote.		
20018	OrigQuoteMsgID	String	N
	Reference to previous QuoteMsgID in case of modification. Custom field.		
	component block <QuoteGrp>		

3.10.3. 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.

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 the value in the request message
- to reject a *Quote* message, with QuoteMsgID set to 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 the value in the request message
- to reject a *QuoteCancel* message, with QuoteStatus set to 5 (Rejected) and QuoteMsgID set to the value in the request message
- in reply to a *QuoteStatusRequest* message, with QuoteStatus set to 8 (Query) and QuoteStatusReqID set to the value in the request message

Tag	Field Name	Type	Req
	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* *7=Expired* *8=Query* *17=Canceled* *21=Traded* *22=Traded and removed (both sides)*		
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 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*		
378	ExecRestatementReason	uint32	N
	Reason for a Quote Status Report sent when communicating an Expired (7) or unsolicited cancel. Field added. *0=GT corporate action* *1=GT renewal/restatement (no corporate action)*		

Tag	Field Name	Type	Req
	12=Cancel on connection loss *99=Other* *100=Book cleared* *101=Volatility guard* *102=Cancel because of changed trading rules* *103=Cancel because of security status change (e.g. Distribution, BuyBack)* *104=Cancel because of quote validation timeout* *105=Cancel because of quote on demand not filled* *106=Expired because not filled at auction* *107=System state changed* *108=Canceled by a user other than the owner of the order or quote* *109=Canceled in immediate mode (Distribution, KnockOutBuyBack), or when IOC or FoK is canceled*		
636	WorkingIndicator	char	N
	Indicates if the quote is currently being worked. Applicable when QuoteType is not 4. Absence means 'Y'. 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.		
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	char	N
	Indicates that this is the last report which will be returned as a result of the request. Field added. *'N'=Not Last Message* *'Y'=Last Message*		
58	Text	String	N
	Error message.		

3.10.4. Quote Cancel (Z)

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

QuoteCancel:

- is replied to with a *QuoteStatusReport* message, with QuoteStatus set to 4 (CanceledAll) or 17 (Canceled) and QuoteMsgID set to the value in the request message
- can be rejected with a *QuoteStatusReport* message, with QuoteStatus set to 5 (Rejected) and QuoteMsgID set to 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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
	component block <SecurityRef>		
131	QuoteReqID	String	N
117	QuoteID	String	N
	Quote identifier assigned by the exchange.		
1166	QuoteMsgID	String	Y
	Unique client-assigned identifier for the request.		
20018	OrigQuoteMsgID	String	N
	Reference to previous QuoteMsgID. Custom field.		
298	QuoteCancelType	uint32	Y
	Identifies the type of quote cancel. *1=Cancel for a security* *4=Cancel all quotes (Does not guarantee message ordering of the resulting deletes and new quotes before operation has completed)* *5=Cancel quote specified in QuoteID or OrigQuoteMsgID*		
60	TransactTime	UTCTimestampNanos	N
	When this quote was cancelled.		

3.10.5. 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 and an update (or cancellation) of the quote is required, or when a new quote is requested for quote on demand.

Tag	Field Name	Type	Req
	component block <StandardHeader>		
	component block <SecurityRef>		
131	QuoteReqID	String	Y
	Unique identifier for quote request.		
117	QuoteID	String	N
	Quote identifier.		
1166	QuoteMsgID	String	N
	Unique client-assigned identifier		
54	Side	char	N

Tag	Field Name	Type	Req
	This is from the perspective of the initiator. Applicable for quote on demand (QOD). *'1'=buy* *'2'=sell*		
38	OrderQty	decimal	N
	Applicable for quote on demand (QOD).		
60	TransactTime	UTCTimestampNanos	N
	The time the QuoteRequest was issued.		

3.10.6. 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 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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
649	QuoteStatusReqID	String	N
263	SubscriptionRequestType	char	Y
	'0'=Snapshot		

3.11. 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 One-party report), is the only model accepted for *non-internal* trades. For internal trades, where the counterparty is the same as the reporting party, the two-party report trading model (figure Two-party report) must be used. 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 following two party roles identifies the trader groups on each side in the trade:

- Buyer/Seller (27) - Used on the trade side you have rights to see trade details on.
- Contra Firm (17) - Used on the trade side without trade details.

Note

A party that has the right to see trade details of both sides, e.g. internal trades, will only receive a single *Trade Capture Report* with both sides. I.e. with Buyer/Seller on both sides.

The exception to this rule is when the trade sides belong to different trader groups, e.g. on-behalf-of accounts.

In the one-party for pass-through model the initiator can cancel the trade as long as it is not confirmed by the counterparty.

Figure 3. Privately negotiated trade, one-party report for pass-through to counterparty.

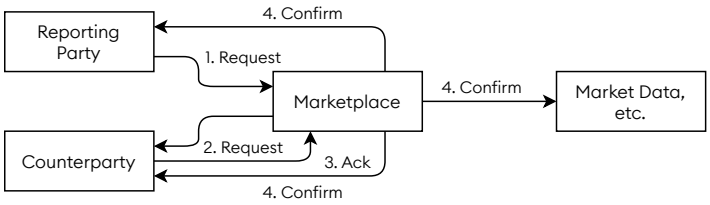
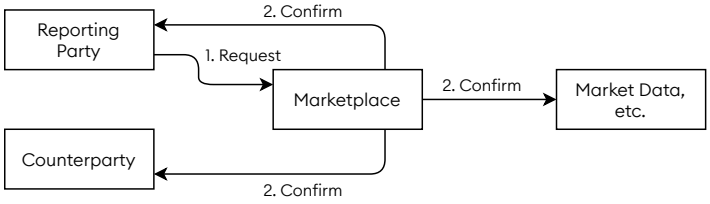


Figure 4. Privately negotiated trade, two-party report.



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. Also see the message flows for one-party report for pass-thru and two-party report.

New automatically matched trade from marketplace. Also used in snapshot response.	**Trade Capture Report** TradeReportTransType = New (0) TradeReportType = Submit (0) TradeHandlingInstr = Trade Confirm ('0') TradeReportID=<marketplace's #1> TradeID=<assigned> MatchStatus = Affirmed ('0')
Cancel trade from marketplace. Also used in snapshot response, but without TradeReportRefID.	**Trade Capture Report** TradeReportTransType = Cancel (1) TradeReportType = Trade Report Cancel (6) TradeHandlingInstr = Trade Confirm ('0') TradeReportID=<marketplace's #2> TradeReportRefID=<marketplace's #1> TradeID=<same> MatchStatus = Affirmed ('0')

For trade snapshots, the following different messages may be sent in response to a *Trade Capture Report Request* message.

Confirmed trade.	**Trade Capture Report** TradeReportTransType = New (0)

	TradeReportType = Submit (0) TradeHandlingInstr = Trade Confirm ('0') TradeReportID=<marketplace's> TradeID=<assigned> MatchStatus = Affirmed ('0')
Canceled previously confirmed trade.	**Trade Capture Report** TradeReportTransType = New (0) TradeReportType = Trade Report Cancel (6) TradeHandlingInstr = Trade Confirm ('0') TradeReportID=<marketplace's> TradeID=<assigned> MatchStatus = Affirmed ('0')

3.11.1. 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=<initiator's #1>
← 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 #1>
← Marketplace forward of submit to counterparty. Also used in snapshot response, to both counterparty and initiator. For initiator, the TradeReportID is <initiator's>.	**Trade Capture Report** TradeReportTransType = New (0) TradeReportType = Alleged (1) TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportID=<marketplace's #1> TradeID=<assigned> MatchStatus = Unaffirmed ('1')
→ Initiator 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 #1> TradeReportID=<initiator's #2>
← Ack from marketplace of initiator cancel.	**Trade Capture Report Ack** TradeReportTransType = Cancel (1) TradeReportType = Submit (0) TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportRefID=<initiator's #1> TradeReportID=<initiator's #2> TradeID=<assigned>
← 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 #1> TradeReportID=<marketplaces' #2> TradeID=<assigned> 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 #1> TradeReportID=<counterparty's #1>
← Ack from marketplace of counterparty decline.	**Trade Capture Report Ack** TradeReportTransType = Replace (2) TradeReportType = Decline (3) TradeHandlingInstr = One-Party Report for Pass-Thru ('3') TradeReportRefID=<marketplace's #1> TradeReportID=<counterparty's #1> TradeID=<assigned>
← 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 #1> TradeReportID=<marketplace's #1> TradeID=<assigned> 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 #1> or <counterparty's #1> TradeReportID=<marketplace's #2> TradeID=<assigned> MatchStatus = Affirmed ('0')
← Reject from marketplace in response to 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 #1> or <marketplace's #1> TradeReportID=<marketplace's #2> TradeID=<assigned> MatchStatus = Unaffirmed ('1')

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

3.11.3. Trade Component Block

This component block is used to define a trade.

Tag	Field Name	Type	Req
1003	TradeID	String	N
	Assigned by the marketplace.		
487	TradeReportTransType	uint32	N
	Transaction type. *0=New* *1=Cancel* *2=Replace*		
856	TradeReportType	uint32	N
	0=Submit *1=Alleged* *2=Accept* *3=Decline*		
	6=Trade Report Cancel		
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	TradePriceCondition	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	OrderCategory	char	N
	Applies only to manual trades. MiFID II regulatory field. *'3'=Privately Negotiated Trade*		
2668	NoTrdRegPublications	Sequence	N
	Applies only to manual trades. MiFID II regulatory field.		
2669	→TrdRegPublicationType	uint32	Y
	0=Pre-trade transparency waiver *1=Post-trade deferral*		
2670	→TrdRegPublicationReason	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".* *6=Deferral due to Large in Scale.*		
1123	TradeHandlingInstr	char	N
	'0'=Trade Confirmation *'1'=Two-Party Report* *'3'=One-Party Report for Pass Through*		
32	LastQty	decimal	N
	Trade quantity of this (last) fill.		
31	LastPx	decimal	N
	Trade price of this (last) fill.		
15	Currency	String	N
	ISO 4217 currency code for the trade. Only used outbound, ignored inbound.		
30	LastMkt	String	N
	Market of execution for last fill. ISO 10383 (MIC). Only used outbound, ignored inbound		
60	TransactTime	UTCTimestampNanos	N

Tag	Field Name	Type	Req
	When this transaction occurred. Execution time of trade or cancellation.		
483	TransBkdTime	UTCTime-stampNanos	N
	When this trade was booked, if other than Transact-Time. Used for manual trade reports and for trade cancellations. Field added.		
573	MatchStatus	char	N
	The status of this trade with respect to matching or comparison. *'0'=Compared, matched or affirmed* *'1'=Uncompared, unmatched, or unaffirmed*		
574	MatchType	char	N
	'1'=One-Party Trade Report (privately negotiated trade) *'2'=Two-Party Trade Report (privately negotiated trade)* *'4'=Auto-match* *'7'=Call Auction*		
277	TradeCondition	MultipleString-Value	N
	Trade conditions set by exchange. 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"=Volatility guard dynamic auction Trade* *"XAS"=Volatility guard static auction Trade* *"XAP"=Order protection auction Trade* *"XAR"=Missing reference price auction trade* *"XLI"=Large In Scale trade* *"0"=Cancel (only used in snapshot)* *"XQ"=Quote on demand trade.*		
552	NoSides	Sequence	N
54	→Side	char	Y
	'1'=buy *'2'=sell*		
37	→OrderID	String	N
20028	→OrderPriority	uint64	N
	Indicates the priority of the order in the orderbook in comparison to other orders on the same level. Higher value means lower priority. Custom field.		
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	char	N
	Defines the lot type assigned to the order. *'1'=Odd Lot* *'2'=Round Lot*		
1057	→AggressorIndicator	char	N
	Used to identify whether the order initiator is an aggressor or not in the trade, or is the report initiator for a one-party for pass-thru trade. *'Y'=Order initiator is aggressor* *'N'=Order initiator is passive*		
528	→OrderCapacity	char	N
	Designates the capacity of the firm placing the order. Absence means 'R' for trades reported to the exchange. *'P'=Principal (DEAL = Deal)* *'R'=Riskless principal (MTCH = Matched)* *'A'=Agency (AOTC = Any Other Capacity)*		
529	→OrderRestrictions	MultipleChar-Value	N
	Restrictions associated with an order. *'B'=Issuer Holding* *'C'=Issue Price Stabilization*		
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*		
453	→NoPartyIDs	Sequence	N
	→→component block <PartyID>		
2593	→NoOrderAttributes	Sequence	N
	→→component block <OrderAttribute>		

3.11.4. 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 the value in the request message
- is replied to with a *TradeCaptureReportAck* message, with TradeReportRejectReason set to 0 (Successful) and TradeReportID set to the value in the request message
- can be rejected with a *TradeCaptureReportAck* message, with TradeReportRejectReason set to the reject reason and TradeReportID set to 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 the value in the request message
- in reply to a *TradeCaptureReportRequest* message, with TradeRequestID set to the value in the request message

Tag	Field Name	Type	Req
	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 cancellation.		
568	TradeRequestID	String	N
	Request ID if this message is in response to a Trade Capture Report Request.		
912	LastRptRequested	char	N
	Indicates that this is the last report which will be returned as a result of the request. *'N'=Not Last Message* *'Y'=Last Message*		
	component block <SecurityRef>		
454	NoSecurityAltID	Sequence	N
455	→SecurityAltID	String	Y
	Alternative security identifier of type specified in SecurityAltIDSource.		
456	→SecurityAltIDSource	char	Y
	Identifies the class of SecurityID. *'M'=Marketplace-assigned identifier* *'4'=ISIN* *'8'=Exchange Symbol* *'D'=Valoren*		
	component block <Trade>		

3.11.5. 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 the value in the request message
- to reject a *TradeCaptureReport* message, with TradeReportRejectReason set to the reject reason and TradeReportID set to the value in the request message

Tag	Field Name	Type	Req
	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 cancellation.		
568	TradeRequestID	String	N
	Request ID if this message is in response to a Trade Capture Report Request.		
912	LastRptRequested	char	N
	Indicates that this is the last report which will be returned as a result of the request. *'N'=Not Last Message* *'Y'=Last Message*		
751	TradeReportRejectReason	uint32	N
	0=Successful (default) *1=Invalid party information* *2=Unknown instrument* *3=Unauthorized to report trades* *4=Invalid trade type* *99=Other* *100=Manual trades not allowed in any knockout state* *101=Duplicate TradeReportID* *102=Manual trades are not allowed for this instrument*		
	component block <SecurityRef>		
	component block <Trade>		
58	Text	String	N
	Error message.		

3.11.6. 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). At least the 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 the value in the request message

- is replied to with a *TradeCaptureReportRequestAck* message, with TradeRequestResult set to 0 (Successful) and TradeRequestID set to the value in the request message
- can be rejected with a *TradeCaptureReportRequestAck* message, with TradeRequestResult set to the reject reason and TradeRequestID set to 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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
568	TradeRequestID	String	Y
	Identifier for the trade request.		
569	TradeRequestType	uint32	Y
	0=All trades (last e.g. 72 hours) *1=Matched trades matching criteria provided on request*		
580	NoDates	Sequence	N
	Range of dates. Since (NoDates=1) or Between (NoDates=2) dates, inclusive.		
60	→TransactTime	UTCTimestampNanos	Y
	When the trade was created.		

3.11.7. 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 the value in the request message
- to reject a *TradeCaptureReportRequest* message, with TradeRequestResult set to the reject reason and TradeRequestID set to the value in the request message

Tag	Field Name	Type	Req
	component block <StandardHeader>		
568	TradeRequestID	String	Y
	Identifier for the trade request.		
569	TradeRequestType	uint32	Y
	0=All trades (last e.g. 72 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.		

3.12. Security Status Messages

3.12.1. User Security Status Update Request (FU)

The *User Security Status Update Request* message allows a member with sufficient rights to change the security status sub-state of a specific instrument. If the request is accepted, the new security status sub-state will be published by a *Security Status* message on the market data service.

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 the value in the request message
- can be rejected with an *UserSecurityStatusUpdateResponse* message, with FinancialStatusResult set to the reject reason and SecurityStatusUpdateRequestID set to 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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
	component block <SecurityRef>		
20040	SecurityStatusUpdateRequestID	String	Y
20049	NoUpdates	Sequence	N
20038	→FinancialStatusUpdateType	uint32	Y
	Financial status type. *1=Knock instrument (will result in knockout or knockout buyback)* *3=Buyback* *4=Distribution* *6=Recalculated*		

Tag	Field Name	Type	Req
20050	→FinancialStatusUpdate-Value	uint32	Y
	Financial status operation. *1=Enable* *2=Clear*		

3.12.2. User Security Status Update Response (FR)

UserSecurityStatusUpdateResponse is sent:

- in reply to an *UserSecurityStatusUpdateRequest* message, with SecurityStatusUpdateRequestID set to 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 the value in the request message

Tag	Field Name	Type	Req
	component block <StandardHeader>		
	component block <SecurityRef>		
20040	SecurityStatusUp-dateRequestID	String	Y
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		

3.13. Quote Validation

The quote validation mechanism can be enabled for one market maker at a time for a security. Only one quote with quote validation is allowed per security at any given time, and is used by setting *QuoteType* to 4 (*Initially Tradable*).

When the security is in continuous trading (open), and an order is entered for a security with the quote validation mechanism enabled, one of the following actions is taken:

1. If the order would result in a match (trade) with a quote from the market maker. → Put the order in a queue.
2. If there already are other orders in the queue. → Put the order in a queue (regardless if it would match the quote with quote validation).
3. Otherwise. → Same as without quote validation, i.e. match the order against any other orders in the order book and put the remaining volume in the order book of the security.

Orders that are placed in the queue are accepted but not executed nor visible in the market data. This is reflected in the *Execution Report* by having *WorkingIndicator* set to N (*Not Yet In Working State*). Orders that are deleted are removed from the queue immediately. An order in the queue that is modified will be moved to the end of the queue if the modification would cause the order to lose priority, otherwise the order will keep its place in the queue.

Immediately when an order is inserted into an empty queue a *Quote Request* message is sent to the market maker, indicating that a trade is imminent. Notice that no information about the order (price, type or volume) is given to the market maker. The market maker must reply to the *Quote Request* as fast as possible, within a specified time period (default 600 ms). If no answer arrives within this period the quote is removed from the order book.

The quote update is matched against the order book before the queue, this is because the update is modelled as occurring exactly before the first order was placed in the queue.

If the quote is removed, then all order operations in the queue are simply executed.

A quote update that is not a direct response to a *Quote Request* while awaiting a response, will be rejected. This way a market maker cannot accidentally accept a *Quote Request*. Once the reply is received or the timeout has been reached, spontaneous quote updates will be accepted again.

3.14. Quote on Demand

Quote on demand is a mechanism where an order can initiate a private auction, separate from the central limit order book matching. A *Quote Request* is sent to the market makers for the security, and only the order quantity is revealed by default (side is not revealed by default). Before the automatic auction ends (default 1 second) the market makers must reply with a *Quote* to participate. The order is locked throughout the auction, while quotes may be continuously updated.

At the auction uncross, only the order can match against the market maker quotes, i.e. quotes does not match against each other. If the entire order volume can be matched, trade(s) will be disseminated and any remaining quotes canceled. Otherwise, the order and any quotes are canceled.

Optionally, the order may have a sweep order book instruction. In this case, the order will also match against the central limit order book at the uncross time.

To initiate an automatic quote on demand auction, the *AuctionType* set to 100 (*QodAuto*) in the order, and *TimeInForce* must be to B (*Good for Auction*). The sweep order book instruction is activated by setting *ExecInst* to 'd' (*Sweep Order Book*).

> **Note**
>
> Since the order is locked throughout the duration of the automatic auction, the *ExecInst* value 'o' (*Cancel on Connection Loss*) is not allowed here.

4. Market Data Service

The market data 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, "Market Definition Request (BT)".

Trading Session Status — See the *Trading Session Status Request* message in Section 4.5.6, "Trading Session Status Request (g)".

Securities — See the *Security List Request* message in Section 4.4.2, "Security List Request (x)".

Security Status — See the *Security Mass Status Request* message in Section 4.4.5, "Security Mass Status Request (CN)".

Market Data — See the *Market Data Request* message in Section 4.6.2, "Market Data Request (V)".

Corporate Actions — See the *Corporate Action Request* message in Section 4.7.3, "Corporate Action Request (U2)".

4.2. Message Overview

The following messages can be sent/received by the client to/from the market data service. Depending on the role 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 2. Message overview.

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

The following are examples of roles 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, Securities=read-only, Corporate Actions=none, Trading Session Status=none, Security Status=none, Market Data=none.*

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, Securities=read-only, Corporate Actions=read-only, Trading Session Status=read-only, Security Status=read-only, Market Data=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.

Tag	Field Name	Type	Req
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, the value "ZZ" to specify physical ownership of the security (e.g. stock certificate), or a value beginning with "DLT-" to identify a distributed ledger network.		
40471	BusinessCenter	String	N

Tag	Field Name	Type	Req
	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.

Tag	Field Name	Type	Req
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)		
2349	PricePrecision	int32	N
	Specifies the price decimal precision, including for price type percentage (for example, 99.789% has three decimals). Field added.		
20054	MaxOrderExpireDuration	uint32	N
	Max duration in seconds of ExpireTime in GTC orders. Custom field.		
20055	MaxTradeTransBkd-TimeDiff	uint32	N
	Max time difference in seconds between Transact-Time and TransBkdTime of trades, i.e. how far back in time a manual trade can be reported. Custom field.		
1787	RefTickTableID	uint32	N
	The ID of the tick rules table used for the instrument. Field added.		
1205	NoTickRules	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	→StartTickPriceRange	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	NoMatchRules	Sequence	N
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	char	N
	The point in the matching process at which the matching algorithm applies. '1'=One-Party Trade Report (privately negotiated trade) '2'=Two-Party Trade Report (privately negotiated trade) '4'=Auto-match '7'=Call Auction		
20057	MarketOrderRules	MultipleChar-Value	N
	The rules that applies for market order. Custom field. '0'=No values '1'=Allow instantaneous (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 instantaneous (IOC or FoK) market orders. '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	OrderProtectionAuction-TimeMin	uint32	N
	Lower bound in milliseconds of duration of the order protection auction. Custom field.		
20059	OrderProtectionAuction-TimeMax	uint32	N
	Upper bound in milliseconds of duration of the order protection auction. Custom field.		
20067	MissingReferen-cePriceAuctionTimeMin	uint32	N
	Lower bound in milliseconds of duration of the missing reference price auction. Custom field.		
20068	MissingReferen-cePriceAuctionTimeMax	uint32	N
	Upper bound in milliseconds of duration of the missing reference price auction. Custom field.		
20052	AllowReserveOrder	char	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	MinReserveOrderValue	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	MinReserveOrderValue-Currency	String	N
	Currency for MinReserveOrderValue. ISO 4217 currency code. Custom field.		
20062	MarketDataRules	MultipleChar-Value	N

Tag	Field Name	Type	Req
	Market data visibility rules. Custom field. *'0'=No values* *'1'=Reveal counterparty information for orders and trades* *'2'=Distribute orders during Pre-Open* *'3'=Distribute equilibrium price during auctions*		
20064	PartyRules	MultipleChar-Value	N
	Party information rules that applies. Custom field. *'0'=No values* *'1'=Executing trader is required for orders and quotes.* *'2'=ClientID is required for orders.* *'3'=ClientID is NOT permitted for quotes.*		
20066	TradeReportRules	MultipleChar-Value	N
	Rules for manual trade reports. Custom field. *'0'=No values* *'1'=Allow all trade reports.*		

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 specified as `Price=99.5`.

Tag	Field Name	Type	Req
	component block <SecurityRef>		
454	NoSecurityAltID	Sequence	N
455	→SecurityAltID	String	Y
	Alternative security identifier of type specified in SecurityAltIDSource.		
456	→SecurityAltIDSource	char	Y
	Identifies the class of SecurityID. *'M'=Marketplace-assigned identifier* *'4'=ISIN* *'8'=Exchange Symbol* *'D'=Valoren*		
	component block <SecurityDefaults>		
1310	NoMarketSegments	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>		
1184	SecurityXMLLen	Length	N
1185	SecurityXML	UnicodeString	
	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*		
20071	LargeInScaleThreshold	decimal	N
	Amount greater than or equal to thís threshold qualify as Large In Scale. Custom Field.		

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 the value in the request message
- can be rejected with a *SecurityList* message, with SecurityRequestResult set to the reject reason and SecurityReqID set to 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

Tag	Field Name	Type	Req
	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 the value in the request message
- to reject a *SecurityListRequest* message, with SecurityRequestResult set to the reject reason and SecurityReqID set to the value in the request message

Tag	Field Name	Type	Req
	component block <StandardHeader>		

Tag	Field Name	Type	Req
320	SecurityReqID	String	N
560	SecurityRequestResult	uint32	N
	0=Valid request (default) *1=Invalid or unsupported request*		
893	LastFragment	char	N
	Indicates whether this is the last fragment in a sequence of message fragments. *'N'=Not Last Message* *'Y'=Last Message*		
146	NoRelatedSym	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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
980	SecurityUpdateAction	char	N
	'A'=Add *'D'=Delete* *'M'=Modify*		
20027	SecurityMoveIndicator	char	N
	Absence 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 (CN)

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 the value in the request message
- can be rejected with a *BusinessMessageReject* message, with BusinessRejectReason set to the reject reason and RefMsgType set to CN
- 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

Tag	Field Name	Type	Req
	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.

Tag	Field Name	Type	Req
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* *103=Scheduled auction*		
327	HaltReason	uint32	N
	Denotes the reason for the Opening Delay or Trading Halt. *100=Regulatory Halt* *101=Other*		
292	CorporateAction	MultipleString-Value	N
	"A"=Ex-Dividend *"C"=Ex-Rights* *"I"=Reverse Stock Split* *"J"=Standard-Integer Stock Split* *"Q"=Tender Offer*		
291	FinancialStatus	MultipleString-Value	N
	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* *"U"=Buyback* *"V"=Distribution* *"Z"=Under observation* *"D"=Volatility guard dynamic* *"S"=Volatility guard static* *"M"=Order protection mode* *"P"=Order protection auction* *"C"=Recalculated* *"R"=Missing reference price auction* *"G"=Generic (unscheduled) auction*		

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 the value in the request message

Tag	Field Name	Type	Req
	component block <StandardHeader>		
324	SecurityStatusReqID	String	N
912	LastRptRequested	char	N
	Indicates that this is the last report which will be returned as a result of the request. Field added. *'N'=Not Last Message* *'Y'=Last Message*		
	component block <SecurityRef>		
	component block <SecurityStat>		
60	TransactTime	UTCTimestampNanos	N
	When the scurity was last modified.		

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, which is used to convey the status of the market segment.

The market status is conveyed using the *Trading Session Status* message. The status of each security is sent individually using the *Security Status* message. The timing between the market status and the security status is not perfect, especially in the case of randomized opening of the market. This means that the security status should be used to see if e.g. the security is open for trading, and the market 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.

Tag	Field Name	Type	Req
1301	MarketID	String	Y
	ISO 10383 Market Identifier Code (MIC).		
1300	MarketSegmentID	String	N
	Identifies the market segment.		
1396	MarketSegmentDesc	String	N
	Description or name of market segment.		
1397	EncodedMktSegmDescLen	Length	N
1398	EncodedMktSegmDesc	UnicodeString	
	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 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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
1393	MarketReqID	String	Y
	Unique request id.		
263	SubscriptionRequestType	char	Y
	'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 the value in the request message

Tag	Field Name	Type	Req
	component block <StandardHeader>		
1393	MarketReqID	String	N
	Reference to the request.		
912	LastRptRequested	char	N
	Indicates that this is the last report which will be returned as a result of the request. 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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
1394	MarketReportID	String	Y
	Unique identifier for each MarketDefinitionUpdateReport message.		
1395	MarketUpdateAction	char	N
	'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.

Tag	Field Name	Type	Req
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		
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* *9=Scheduled auction*		
912	LastRptRequested	char	N
	Indicates that this is the last message which will be returned as a result of the request. *Field added.* *'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 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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
335	TradSesReqID	String	Y
	Unique request id.		
263	SubscriptionRequestType	char	Y
	'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 the value in the request message

Tag	Field Name	Type	Req
	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. Market orders do not have a price.

Offer ('1') *MDEntryPx* and *MDEntrySize* contains the price and volume of the offer order or quote. Market orders do not have a price.

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

The statistics are maintained for *session* and *day*. The values can be requested in a snapshot until they are generated or cleared next time.

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 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 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 order book of the instrument.

Opening statistics for the *day session* is defined as the first opening of any session and the last closing taken from the last session. Session and day values are differentiated by the *MDStatsScope* field.

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 be 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 *volatility guard auction*, the equilibrium price, available bid and ask volume are continuously disseminated during and upon entry of the auction for each order book. The equilibrium price with available buy and sell volume are updated every time there is a change in an order book but no more than once per second per order book. In the case where an order book 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 order book.

Equilibrium Buy ('b') If the order book 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 order book 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.

Tag	Field Name	Type	Req
269	MDEntryType	char	Y
	'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%).*		
20016	MDStatScope	uint32	N
	Defines the scope of the statistics in periods of time. *Custom field.* *1=Session* *2=Day*		
270	MDEntryPx	decimal	N
	Entry price.		
271	MDEntrySize	decimal	N
	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.		
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	char	N
	Match type for trades. *'1'=One-Party Trade Report (privately negotiated trade)* *'2'=Two-Party Trade Report (privately negotiated trade)* *'4'=Auto-match* *'7'=Call Auction*		
828	TrdType	uint32	N
	Trade type for trades. *0=Regular Trade* *52=Exchange Granted Trade*		
855	SecondaryTrdType	uint32	N
	Type of trade assigned to a trade. Used in addition to TrdType(828) *0=Regular Trade.* *64=Benchmark Trade.*		
277	TradeCondition	MultipleString-Value	N
	Trade conditions set by exchange. *"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"=Volatility guard dynamic auction Trade* *"XAS"=Volatility guard static auction Trade* *"XAP"=Order protection auction Trade* *"XAR"=Missing reference price auction trade* *"XLI"=Large In Scale trade* *"0"=Cancel (only used in snapshot)* *"XQ"=Quote on demand trade.*		
1839	TradePriceCondition	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	AlgorithmicTradeIndicator	uint32	N
	MiFID II regulatory field. Absence means '0'. *0=Non-algorithmic trade* *1=Algorithmic trade*		
1115	OrderCategory	char	N
	Applies only to manual trades. MiFID II regulatory field. *'3'=Privately Negotiated Trade*		
2668	NoTrdRegPublications	Sequence	N
	Applies only to manual trades. MiFID II regulatory field.		
2669	→TrdRegPublicationType	uint32	Y
	0=Pre-trade transparency waiver *1=Post-trade deferral*		

Tag	Field Name	Type	Req
2670	→TrdRegPublicationReason	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".* *6=Deferral due to Large in Scale.*		
1093	LotType	char	N
	Defines the lot type assigned to the order. *'1'=Odd Lot* *'2'=Round Lot*		
60	TransactTime	UTCTimestampNanos	N
	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).		
483	TransBkdTime	UTCTimestampNanos	N
	When the trade was booked, if other than TransactTime. Used for manual trade reports. Field added (partially).		
5797	AggressorSide	char	N
	Indicates which side is aggressor of the trade. If there is no value present, then there is no aggressor. Custom field. *'1'=buy* *'2'=sell*		
20033	MDEntryParticipantType	uint32	N
	Indicates which type of market participant this MDEntry originates from. Only applicable if MDEntryType = '0', '1' or '5'. Custom field. Absence means 0. *0=Ordinary participant* *1=Market Maker* *2=Liquidity Provider*		
1024	MDOriginType	uint32	N
	0=Book (Central limit order book) *1=Off-Book* *3=Quote driven market* *4=Dark order book* *5=Markets where matching occurs only in scheduled auctions* *6=Discretionary quoting on request or "request for quote" market* *8=Hybrid market* *9=Other market*		
336	TradingSessionID	String	N
	Identifier for a trading session. *"1"=1 = Day*		
625	TradingSessionSubID	String	N
	A trading phase within a trading session. *"1"=1 = Pre trading* *"2"=2 = Opening or opening auction* *"3"=3 = (Continuous) Trading* *"4"=4 = Closing or closing auction* *"5"=5 = Post trading* *"9"=9 = Unscheduled intraday auction. An unscheduled intraday auction might be triggered by a circuit breaker.* *"14"=14 = Order initiated auction*		

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. At least the 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 the value in the request message
- can be rejected with a *MarketDataRequestReject* message, with MDReqRejReason set to the reject reason and MDReqID set to 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

Tag	Field Name	Type	Req
	component block <StandardHeader>		
262	MDReqID	String	Y
	Unique identifier for Market Data Request.		
263	SubscriptionRequestType	char	Y
	'0'=Snapshot		
264	MarketDepth	uint32	Y
	0=Full book		
267	NoMDEntryTypes	Sequence	Y
	Requested entry types. Empty list means all entry types.		
269	→MDEntryType	char	Y
	'0'=Bid *'1'=Offer*		

Tag	Field Name	Type	Req
	'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%).*		
580	NoDates	Sequence	N
	Range of dates for requested trades. Since (NoDates=1) or Between (NoDates=2) dates, inclusive. Sequence added.		
60	→TransactTime	UTCTimestampNanos	Y
	When the trade was created.		

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 the value in the request message

Tag	Field Name	Type	Req
	component block <StandardHeader>		
262	MDReqID	String	N
	component block <SecurityRef>		
268	NoMDEntries	Sequence	Y
	→component block <MDEntry>		
912	LastRptRequested	char	N
	Indicates that this is the last report which will be returned as a result of the request. Field added. *'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.

Tag	Field Name	Type	Req
	component block <StandardHeader>		
268	NoMDEntries	Sequence	Y
	→component block <MDEntry>		
279	→MDUpdateAction	char	Y
	'0'=New *'1'=Change*		
	'2'=Delete		
	→component block <SecurityRef>		
1175	→NoStatsIndicators	Sequence	N
1176	→→StatsType	uint32	Y
	Type of statistics. *1=Exchange Last* *2=High / Low Price* *3=Average Price (VWAP, TWAP etc)* *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 the value in the request message

Tag	Field Name	Type	Req
	component block <StandardHeader>		
262	MDReqID	String	Y
	Refers to the request.		
281	MDReqRejReason	char	N
	'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.

Tag	Field Name	Type	Req
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

Tag	Field Name	Type	Req
	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*		
230	ExDate	LocalMktDate	N
	When this corporate action takes effect.		
60	TransactTime	UTCTime-stampNanos	N
	When this corporate action was created or updated.		
20006	AdjustmentFactorNu-merator	uint32	N
	The adjustment factor 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.		
20022	AdjustmentFactorDe-nominator	uint32	N
	The adjustment factor 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 the value in the request message
- to reject a *CorporateActionRequest* message, with CorpActionResult set to the reject reason and CorpActionReqID set to the value in the request message

Tag	Field Name	Type	Req
	component block <StandardHeader>		
	component block <SecurityRef>		
20009	CorpActionReqID	String	N
	Unique request identifier. Custom field.		
20012	ActionResult	uint32	N
	Result returned to a Corporate Action Request message. Custom field. *0=Succeeded (default)* *1=Invalid or unsupported request*		
912	LastRptRequested	char	N
	Indicates that this is the last report which will be returned as a result of the request. *'N'=Not Last Message* *'Y'=Last Message*		
20011	CorpUpdateAction	char	N
	The update action of an incremental update. Absent in a snapshot response. 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 the value in the request message
- can be rejected with a *CorporateActionReport* message, with CorpActionResult set to the reject reason and CorpActionReqID set to 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

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

5. MiFID II Regulatory fields

5.1. Post trade transparency

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

- *BENCH*
  - SecondaryTrdType(855) = 64 (Benchmark trade)
- *NPFT*
  - TradePriceCondition(1839) = 15 (Non price forming trade)
- *TNCP*
  - TradePriceCondition(1839) = 16 (Trade not contributing to the price discovery process)
- *SDIV*
  - TradePriceCondition(1839) = 13 (Special dividend trade)
- *ALGO*
  - AlgorithmicTradeIndicator(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)

5.2. Order Record Keeping

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.

Party information is sent using anonymized numerical short code values upon order entry, and later mapped to real identifiers. See the *Elasticia ORK Short Code API* document for more information on requirements and validation rules for party information related to order record keeping.

The field PartyID (448) should contain the short code value. The PartyRole (452) determines the party, where:

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

If a PartyRole is populated in an order or quote, it is required that the accompanying fields PartySourceID, PartyID and PartyRoleQualifier are also populated.

Party information is required on the *first submission* of an order (New Order Single) or a quote. Party information *cannot be changed in subsequent updates*, and may be omitted.