Deutsche Börse AG

Mailing Address
Mergenthalerallee 61
65760 Eschborn

Web
www.deutsche-boerse.com

DFS180 - M7 - Public Message Interface- Prospect Version

M7 Release 6.0.10 – Market Data Team Modification date Aug 2017

Version	1.05
Status	Final
Filename	DFS180 - M7 - Public Message Interface (Schema 6.0) - DRAFT VERSION - v1.docx
Date	07/09/2017
Author	M7 Project Team
Reviewer	M7 Project Manager

Table of Contents

Terminology

Client Program or application acting as a client of the AMQP server.

Trader Person that logs into the client to participate in the market.

Balancing Phase Additional trading phase, which starts after the end of the normal trading phase and ends in relation to delivery start. Normal orders from market participants can be entered but cannot trigger a trade execution. Only balance orders can trigger a trade execution (balance order entry is usually limited to one specific member). The order book can be crossed during the balancing phase.

Version history

0.99 Document created against schema 5.0 document version 2

1 Added risk parameters for delivery areas, risk set messages, implied orders messages.

2 Added external limit identifier and version to the Cash limit report.

1.00 Version created with tracked changes from API 5.0.

1.01 Added remoteOrdrId and remoteTradeId to trade settlement report (TradeStlmntRprt) in chapter 7.2.2.

1.02 Aligned with M7 6.0.10 change requests

removed defaultDlvryAreaId from user entity

Added “broadcast audience” in message summary tables, updated routing keys

Added chapters 5.1.12, 9 and 10

1.03 Header and Footer updated, milliseconds added in chapter 5.1.9, short description changed for field clearingAcctType in chapter 6.2.1., 6.2.4, 6.2.5, 6.3.2, 6.4.13, 7.2.2, field currency set to mendatory in chapter 6.4.7, field dlvryAreaId set to mandatory in chapter 6.4.27, field riskSetId removed from DlvryAreaInfoRprt in chapter 6.6.4, field aggressorIndicator added to TradeStlmntRprt in chapter 7.2.2, several fileds replaced from RiskSetInfoRprt in chapter 6.6.16, field riskSetId removed from chapter 7.3.13, field remoteState added to ModifyDlvryAreaReq in chapter 7.3.14, field version changed to externalVersion in chapter 7.3.15/16, field riskSetId + value ‘CRO’ for field product type added to FullProdInfoRpt in chapter 7.3.19,

1.04 Adjusted chapter 9.14 (migration rules).

Clarified recallGrantedTime in TradeStlmntRprt (chapter 7.2.2) and TradeCaptureRprt (chapter 6.4.13)

1.05 Field bespoke set to optional (chapter 6.4.17)

Usr node added (chapter 6.6.1)

Set all field in buy/sell tags to optional (chapter 6.4.13)

1 Introduction

This document describes the Public Message Interface that the backend system provides to its third party clients. In addition the Administrative Message Interface that the backend system provides to client applications that support users with administrative privileges such as Market Operation, are described. The Administrative Message Interface extends the Public Message Interface to provide additional functionality to these administrative users.

The Public Message Interface allows clients to communicate with the backend system via a programmable interface.

There are 2 kinds of communication between clients and backend:

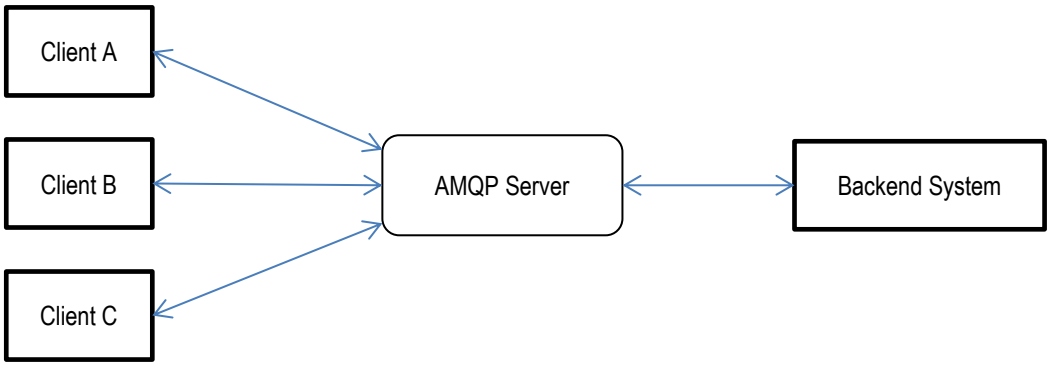
- Management and inquiry requests initiated by the client applications, and the corresponding replies from backend
- Broadcast messages sent by the backend to all or specific clients, triggered by market events (e.g. new orders or trades) or by changes in public or private reference data

The communication with the backend system is based on *Advanced Message Queuing Protocol* (AMQP) as the transport layer. AMQP is a platform- and language-neutral open standard for the wire protocol[1] on the application layer of OSI model.

Payload of the messages sent over AMQP is formatted in XML. See www.w3.org/XML for information about XML.

1.1 Overview

Clients that use the Message Interface communicate with an AMQP server, which in turn communicates with the backend system. The backend system itself is behind firewall and is not directly accessible for the clients.



Overview of communication between clients and the backend system

The AMQP server is currently using the AMQP implementation from RabbitMQ. This version supports AMQP versions 0.9.1, 0.9 and 0.8. See www.amqp.org and www.rabbitmq.com for more information.

For an optimal interoperability, clients should be implemented using an AMQP client library for RabbitMQ, as interoperability limitations are known in the RabbitMQ server implementation, please refer to the following webpage: https://www.rabbitmq.com/interoperability.html.

[1] Wire protocol refers to a way of getting data from point to point when multiple applications needs to interoperate

2 Getting Started

To get started with using the Public Message Interface, you need to:

- Request an account on the AMQP server at the granting authority. You will obtain the following package:
  - A SSL client certificate and password to connect to the AMQP Server (see section Security)
  - A unique application id (if not already exists for the used client implementation) that has to be used for further communications with the backend system
  - The source code of the reference client implementation
  - The message format as XML Schema files (the files are part of the source code for the reference client implementation). You will need these schema files to be able to correctly format messages sent to the backend system and read its responses. See section for details.
- Request a trader account if not already exist that is needed for participating in the market.
- Download AMQP client library for RabbitMQ from www.rabbitmq.com. Libraries from other AMQP vendors are not supported due to interoperability issues (vendor interoperability expected since AMQP version 1.0).
- Implement your client. The way how your program should communicate with the backend system is described in section Message Exchange Patterns.

2.1 AMQP

AMQP (Advanced Message Queuing Protocol) is wire-level protocol that enables message-based communication through the use of an AMQP compliant middleware server (the message broker). In the case of the Public Message Interface this message broker is RabbitMQ.

AMQP defines a modular set of components for the broker as well as standard rules for connecting them. There are 3 main types of components, which are connected into processing chains to create the desired functionality:

- The **"exchange"** receives messages from publisher applications and routes these to "message queues" based on arbitrary criteria, usually message properties or content
- The **"message queue"** stores messages until they can be safely processed by a "consuming" application (or multiple applications)
- The **"binding"** defines the relationship between a message queue and an exchange and provides the message routing criteria

The exchanges, message queues and bindings that are set up by the backend system are described in more detail in the following chapter.

Please refer to the AMQP and RabbitMQ documentation for more details on the use and configuration capabilities of both AMQP and the RabbitMQ client library.

2.1.1 Connecting to the AMQP server

The first step every client program needs to take is to establish a connection to the AMQP server. This connection can then be used to create the channels that are used for communication between the client application and the backend server. The various channels needed to communicate with the backend can all share the same connection. The same holds true for a multithreaded implementation: all threads of a client program typically share the same connection, but channels cannot be shared; each channel can only be used by a single thread.

Creating a connection requires the following information to be specified:

- Username, password, server host, port and virtual host to connect to.
  This information will be supplied by the granting authority.
- SSL context: client certificate and client certificate key

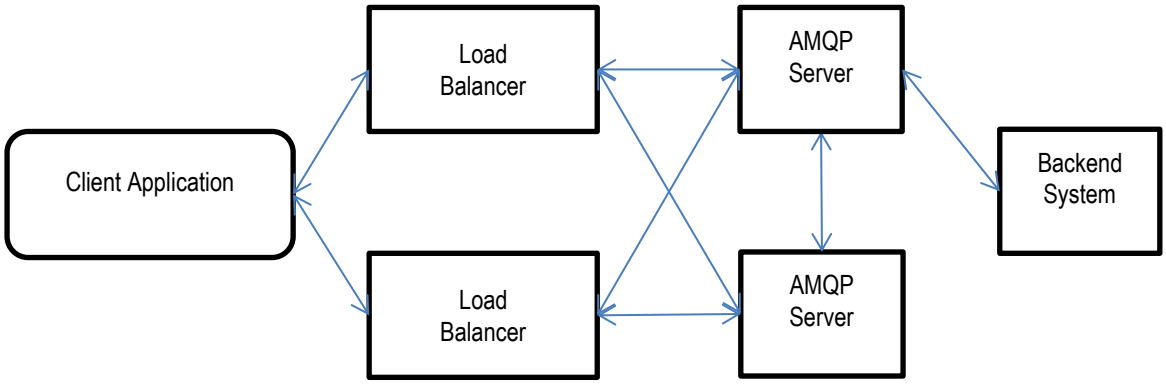
It is mandatory that all connections to the AMQP server are created with the AMQP heartbeats enabled. The recommended heartbeat period is 30-60 seconds. Connections that do not have heartbeats enabled will time out on the firewall in case there is no traffic on the connection for a longer period of time.

In the RabbitMQ Java client, AMQP-level heartbeats are enabled using method `ConnectionFactory.setRequestedHeartbeat(int)`, *specifying the heartbeat interval length in seconds.*

In addition, client programs may want to register a shutdown listener on the connection and/or on channels. This can help the application to detect connection loss faster.

2.1.2 Client Failover

Access to the AMQP server is possible through two different data centres with different IP addresses and URLs. In standard operating phase, both data centres are active and can be used to connect.



The URL or IP address is the only difference between both ways to connect – all other connection details (port, username, password, client certificate) stay the same.

Failover needs to be implemented on client side by using the URLs provided. In case the connection through the first URL is not possible, the client needs to try to connect to the second URL. DBAG generally suggests to implement round-robin within the log-in procedure to minimise operational impact in case one datacentre is temporarily unavailable.

Please notice that independent of the URL / IP address chosen to connect, all commands will always be routed to the master backend instance.

2.2 Application ID

Each client application will receive its own id that has to be used for further communication with the backend system. If no application id is used or the application id is not configured in the backend system the client will not be able to participate in the market. The application id will be verified from the backend system at each request.

The application id will be given to each client by the granting authority. Each client is responsible to keep its id secret.

3 Message Exchange Patterns

Two basic patterns of message exchange between the client and the backend system are supported:

- *Request-response* communication, where client issues a request and waits for a response from the backend system. Each response message is addressed to a specific client (the one that issued the request). This type of communication is initiated by the client.
- *Broadcast* communication, where the backend system publishes notifications that are either public - addressed to all traders - or private - only receivable by privileged traders. Clients may subscribe to receive notification broadcasts they are interested in. This type of communication is initiated by the backend system.

Typically, all market related information is broadcast by the backend system to all the client applications that are authorised to receive that information.

The request – response communication is mainly reserved for “actionable” requests (called “management requests” in the rest of the document) e.g. order entry, trade recall request, etc.

The “inquiry requests” serve to obtain information on the current state of the market or reference data. However, they should only be used at the start of a new session to obtain an initial view of the market or when recovering from communication failures. Subsequent changes in the market situation will be broadcast by the backend system to all connected client applications which should handle these broadcasts to update the market and reference data they present to their users accordingly. Failing to comply with that pattern may lead to revocation of the respective application id. Note that the backend system limits the number of inquiry requests a client application may submit in a period of time to avoid excessive and performance degrading use of the inquiry requests.

The rest of this chapter provides more detail on these 2 modes of communication with the backend system.

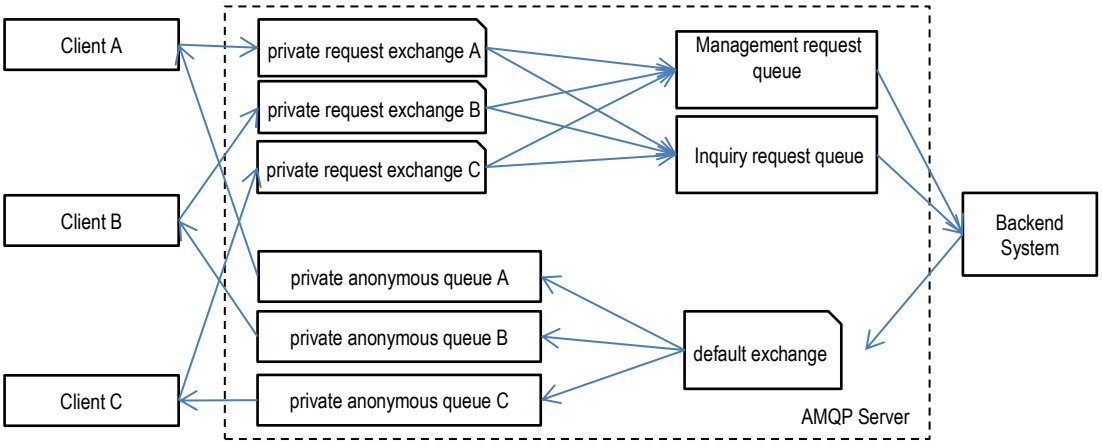
3.1 Request-Response communication

Traders that want to participate in the market send their requests to the backend system. Requests are sent to private trader-specific request exchanges. After processing a request, the backend system sends the response to a private response queue that belongs to the trader that has sent the request.

3.1.1 AMQP configuration

In case a new trader has been setup by the granting authority, the backend system will synchronize with the AMQP Server at runtime and the needed exchanges and queues will be setup within the AMQP Server.

The following diagram details the exchanges and queues that are set up in the AMQP server for request-response based communication between a typical client application and the backend system.



Request-response communication between clients and the backend system

A durable direct exchange named `m7.requestExchange.<login-id>` is set up on the AMQP server for each trader (indicated as "private request exchange X" in the figure above). These exchanges are used for trader requests of all request types.

The response queues used by the client for receiving responses upon requests won't be setup initially by the AMQP Server but from each client. Therefore, the client creates one private response queue and uses the queue name within the reply-to field of a request message. See the reference client source code for looking up the implementation done in Java.

The M7 System guarantees that it will process request messages from a client in the order in which the messages have been delivered to the queue on the AMQP server.

Each client has to send a login request at first using a valid trader that is configured in the M7 System. Without sending a login request a trader cannot participate in the market. As a response of a traders login request a "User" response is send containing all needed information for participating in the market. The login of a client is checked by the M7 System. Although only traders with a specific user role can participate in market via the Public Message Interface. Roles will also be given by the granting authority.

The M7 System supports a single login functionality so any trader can only be logged in once to the M7 System. As a result of this the M7 System may send a "LogoutRprt" broadcast message containing the session id passed to the client within the "User" response after login. This broadcast is send whenever a trader is accessing the M7 System via a different interface or the public Message Interface itself. When receiving a "LogoutRprt" a client has to perform a logout of its trader. If the client doesn't perform a logout of the current trader and allows a parallel login of the same trader both will consume messages from the traders response queue and therefore remove messages.

After receiving a "LogoutRprt" a trader can re-login with forcing the trader with the same login credentials to be logged out

3.1.2 Request Types

The backend system supports two types of requests:

- *Management Requests* are used to enter, modify or delete orders or trades in the backend system.
- *Inquiry Requests* return market or reference data accessible for the client.

As an answer upon a *management request* the backend system will send an "AckResp" to acknowledge the receipt of this request. After processing the request the backend system will send one or more broadcast messages containing the requested change. All these responses are sent to the private response queue of the requesting trader.

If the request results in a change in the market or reference data, a second set of broadcast messages will be sent to all market participants (client applications) notifying them of the change.

Example	*When a client application sends a new order to the backend, it will receive an AckResp in reply to confirm the reception of the order.* *After processing the request, the backend system will send an Order Execution Report to the client application and a Public Order Book Delta Report to all client applications that have access to the market data for the product concerned.*

For details on the content of the requests, see section 5 Message Format.

3.1.2.1 Management-related requests

For management requests, it is possible to send multiple requests of the same type in a single message. The backend system confirms the reception of the request with one AckResp message. Each individual request will be responded by an individual response message.

When sending an order management request, the request may specify a client order id. The response sent by the backend system contains this client order id, to enable the client to relate the response to the order in its own system.

The maximum number of management-related requests bundled in one single request is defined by a limit, which is defined in the XML Schema files for each request. Should this limit be exceeded, the whole request message will be rejected and the

client will get an error response containing information about the current limit setting. The limit may be subject to change in future schema versions.

3.1.2.2 Inquiry requests

The following inquiry request types are supported:

- *Private Messages Requests are* used to retrieve the client specific private messages held by the backend system.
- *Public Messages Requests are* used to retrieve public messages held by the backend system.
- *Trade Requests* are used to retrieve all trades for products a client is assigned to.
- *Order Book Requests* are used to retrieve the current order book situation managed by the backend system.
- *Order Requests* are used to retrieve orders based on the client assignments.
- *Reference Data Requests* are used to retrieve reference data needed by the client for initialization.

For inquiry requests, it is possible to send only a single request in each message. The backend system returns a single response for each inquiry request.

3.1.3 How to Send Requests

A client that wants to send a request to the backend system shall do the following:

- Declare one private (exclusive) response queue with the following name on the AMQP server:

  `m7.private.responseQueue.<login-id>.<unique-id>`

  This queue will be used as long as the client is connected to the broker. Use a *return listener* for the AMQP channel to be able to detect unroutable requests. The unique ID needs to be generated by the client (e.g. UUID, GUID, etc.). The format of the unique ID can be defined by the client, but it's the responsibility of the client to make sure that a second login with the same login id generates a different unique id. Otherwise the AMQP server will refuse a queue creation with the same queue name.

  Max length of queue name is 127 characters and its validated[2] by regexp ^[a-zA-Z0-9-_.:]$
- Create an AMQP message and put the XML-encoded request into the message body.
- Set message attribute `content-type` to the version of XML schema used to encode the message body. See AMQP Message Properties for the formatting of the content-type attribute.
- Set message attribute `reply-to` to the name of the private response queue (`m7.private.responseQueue.<login-id>.<unique-id>`). See AMQP Message Properties for the formatting of the reply-to attribute.
- Set message attribute `user-id` containing the user's login-id. See AMQP Message Properties for the formatting of the user-id attribute
- Set message attribute `app-id` containing the application id given by the backend system Granting Authority. See AMQP Message Properties for the formatting of the app-id attribute
- Set message attribute `correlation-id` to a unique request ID that can be used to match responses with requests. The uniqueness of a correlation-id is the responsibility of each client. The backend system won't check the uniqueness of the correlation-id sent by the clients. See AMQP Message Properties for the formatting of the correlation-id attribute
- Optionally, set message attribute `expiration` to a time stamp when the message should expire (see details below). See AMQP Message Properties for the formatting of the expiration attribute
- Send the message to exchange `m7.requestExchange.<login-id>` in "*Mandatory"* mode with either routing key `m7.request.management` or `m7.request.inquiry` depending on the request type. When

[2] Full reference can be found on https://www.rabbitmq.com/protocol.html

sending requests in mandatory mode the sender will be informed immediately if there is no consumer registered for his request queue. See www.rabbitmq.com for more information.

Once the backend system processed the request, it will send a response to the client's response queue specified in the request's `reply-to` attribute. The response message is sent with attribute `correlation-id` set to the value contained in the request. This allows the client to match responses with requests.

The backend system will send a response to each request. As long as the XML schema version used by the client is supported by the backend system, the response will be encoded using the same schema version that was used for the request.

For details about the request and response formats, see section 5 Message Format.

3.1.3.1 Invalid and Unroutable Requests

If the backend system cannot process a request, because the request is incorrect or cannot be fulfilled, it will still send a negative response. The response message contains details about why the request could not be processed.

If the backend system cannot process the request because the XML schema version in the request message header is missing or invalid, the backend system will send a native error response. This response has set the attribute content-type with the value `x-m7/error`. The body contains an error message encoded in UTF-8. Reasons for sending a native error message may be caused by validation errors detected by the backend system. Validation errors may occur because of

- Application Id not set
- User Id not set
- ContentType not set
- ReplyTo not set
- CorrelationId not set

If the backend system cannot process the request because its XML code is invalid, it will send a special xml error response. See section Message Format for information about the "ErrResp" format.

If the backend system cannot process the request because it is down, the request message will be discarded by the AMQP server and the client will be notified about this via its return listener.

3.1.4 How to Receive Responses

The client must receive the response asynchronously using a *consumer*. The client registers a consumer for the response queue and AMQP invokes the consumer when a message arrives. This method allows the client to wait passively without consuming CPU. The maximum wait time must be limited, because it is possible that the response never arrives (see below).

Once a response is received, the client should extract the following message attributes:

- Attribute `content-type` containing the XML schema version used for encoding of the response. This allows the client to choose the correct XML schema for XML unmarshalling. See AMQP Message Properties for information about the formatting of the content-type attribute.
- Attribute `correlation-id` containing the correlation ID from the request. This allows the client to match responses with requests. See AMQP Message Properties for information about the formatting of the correlation-id attribute.
- Attribute `type` containing the delivered Message classname (e.g. `ContractInfoRprt`) Classname for each message is shown in the Message properties summary table header.

For sample code that receives responses using a consumer, please refer to the reference client implementation.

3.1.5 Acknowledgement of Responses

Client must not leave any unacknowledged messages on AMQP server. Client applications are requested to use auto acknowledgement on RabbitMQ channels to acknowledge the reception of all response messages as soon as the message has been received (before processing of the message).

If client does not receive response in timely manner, it is encouraged to re-inquiry for such message similarly as described in 3.1.7 Unacknowledged Requests.

If server side queue gets filled with unacknowledged messages, it might lead to high memory consumption. The private response queues on server side are constantly monitored and in case one of the queues hit a certain threshold, the connection might be actively disconnected by the server and the Application ID gets deactivated.

Please refer to the Rabbit MQ documentation for more details on the acknowledgement of messages at the AMQP level.

3.1.6 Connection Failure

Clients need to be designed to handle connection closing unexpectedly. When a connection closes unexpectedly some requests will not be acknowledged, the client must re-connect, and then proceed as described in Unacknowledged Requests, below. It is otherwise not possible to know which requests were processed by the backend, and which were discarded when the connection closed.

It is recommended that clients register a shutdown listener for the AMQP connection to implement this behaviour. The shutdown listener will be called whenever the client detects the connection has closed, regardless of the cause.

Clients must use exponential back-off when re-connection repeatedly fails.

3.1.7 Unacknowledged Requests

A request may not be acknowledged because the connection used to send is closed (see 3.1.6 Connection Failure), the request was discarded without being processed, or because the response was discarded. To detect that a message has been discarded, the client must time-out on waiting for responses.

The timeout setting of the client has to take network latency into account. Excessive resending of requests will impact performance.

When a request is unacknowledged the client must determine if the request was received by the backend.

For non-management requests, resend the request to receive the response. You may receive the response more than once. For Management requests, you must send the needed inquiry requests to determine if the management request has been executed. If the request has not executed, re-send the request.

3.1.8 Message Overflow Handling

If the backend system is not able to process requests (it is not listening on the request queue or it is too busy), client attempts to send request messages will be rejected by the AMQP server (because the messages are sent in mandatory mode).

To detect this, the client must register a return listener for the channel being used to send requests. This is the only circumstance under which the client may assume a request was not processed by the backend. See www.rabbitmq.com for more information about how to register return listeners.

This ensures that most requests are either processed immediately or rejected. Otherwise, client requests could get stalled in the request queue without the possibility to cancel them.

3.1.9 Flow Control

If clients would send requests too often, a backlog of unprocessed requests could build up on the server side, resulting in delays in request processing, affecting negatively all clients.

Several measures are taken to prevent this scenario.

3.1.9.1 Request Rate Limit

The backend system constantly monitors the request rate for each user and each inquiry request type. For each inquiry message, the backend system defines a short term and long term request limit per user. Currently the short term and long term time periods are set to 1 minute and 60 minutes respectively.

If the number of a specific inquiry request sent by a user within a time period exceeds the configured limit, the backend system will start rejecting this request from that user, until the request rate goes back below the limit. The backend system will send the following error response to the user:

Limit is " + <requestLimit> + " per " + <requestPeriod> + " ms.

When the request rate is exceeded, instead of processing an incoming request, the backend system sends back a special *back off error response* containing details about the length of the measurement period and the request rate limit.

Note that the limits are specified in such a way as to allow a full initial market state inquiry, as well as a 'normal' amount of failure recovery. If no failures occur, client applications can remain up to date with respect to the market and reference data by processing correctly the broadcast messages sent by the backend system.

The limits are counted starting first request of the particular type.

Example	*If the short and long term limits for a message are set to 1 and 10 respectively, the backend system will only allow one request per minute (application of the short term limit). A subsequent request from the same user should thus be sent at least 1 minute after the previous request of the same message type has been sent.* *In addition, the same request cannot be sent more than 10 times in a period of 1 hour (60 minutes), even if the requests are spaced more than 1 minute apart (application of the long term limit).*

3.1.9.2 Message Expiration

Clients may specify an expiration time for each request message. This allows the clients to protect themselves against the situation, when a request waiting in the request queue for a long time becomes obsolete, but eventually gets processed even if the client does not wish to execute the request anymore.

3.1.9.3 AMQP Server Flow Control

In the AMQP protocol, *flow control* is an emergency procedure used to halt the flow of messages from a peer. It works in the same way between client and server and is implemented by the AMQP `Channel.Flow` command. Flow control is the only mechanism that can stop an over-producing publisher.

This feature is however not used by the RabbitMQ server. Instead, in case the server hits a preconfigured memory watermark, it uses TCP backpressure to temporarily block all connections that publish messages.

The RabbitMQ server persists queue contents to disk, if required. This allows it to keep more messages than fit into the machine memory; the queue size is theoretically only limited by the disk space.

In case of a very high traffic, the AMQP server can be forced to temporarily throttle publishers to gain time to move some queue contents to disk, releasing memory for new messages. Once this is done, the server will start receiving messages again. (Under normal circumstances, this should not happen because the measures described above should always keep the queues relatively small.)

3.1.10 Message type

M7 system uses AMQP message attribute named `type` to provide information about the content of each message. Example: `ContractInfoRprt`

This attribute can be used by client application to determine content of each message even before unpacking/processing.

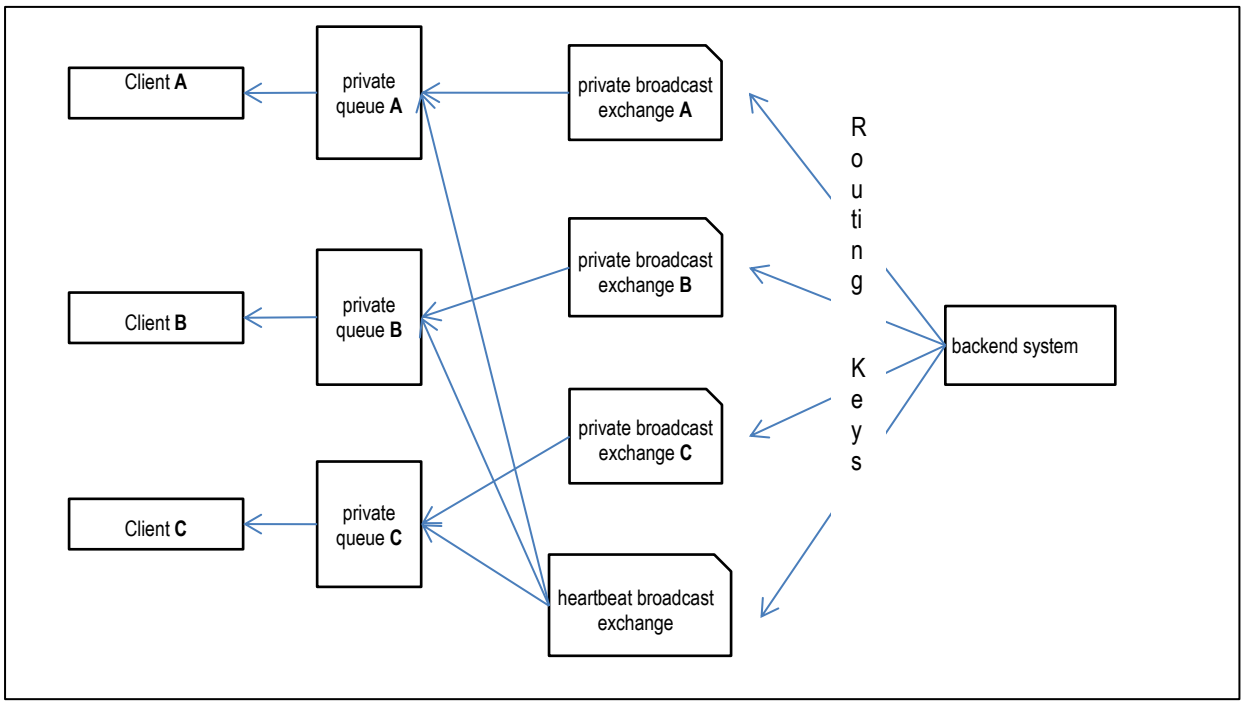
3.2 Broadcast

The backend system broadcasts three types of information:

- *Heartbeat* broadcasts sent in regular intervals allow the clients to monitor the availability of the backend system. This is also used to check if clients are still connected to the backend system.
- *Market Data* broadcasts inform clients about changes within the current market. Traders will receive information only if its assignments are matched by the market data change.
- *Reference Data* broadcasts inform clients about changes on reference data.

Broadcast messages sent by the backend system are distributed (pushed) via the AMQP server to all clients that have subscribed to receive the information.

Several system operations (ex: contract closure, service or delivery areas halt/trading) may generate a large number of broadcasts since they have an impact on reference/market data (ex: contracts or orders state).



Broadcast routing architecture of the backend system

Broadcasts messages are encoded and send using all supported XML schema versions. To avoid supporting too many schema versions and therefore a payload overhead the backend system only supports the latest 2 XML schema versions.

3.2.1 Heartbeat Broadcasts

A durable topic exchange named

- `m7.heartbeatExchange`

is used for broadcasting heartbeat messages. This exchange is bound to the traders broadcast queue.

The heartbeat messages are sent with routing key

```
<schema-version>.m7.heartbeat
```

Each heartbeat message contains information about the heartbeat interval length as well as a message attribute within the header property of each message containing the send timestamp called “server-timestamp”. This allows the clients to monitor

the availability of the backend system and calculate message latency. See Heartbeat section for information about the heartbeat message format.

The backend system has a connection loss functionality to enable a user to specify actions that will be executed in a failure case. If a client has not been connected to the message broker for a specified amount of time the prior defined actions (see Message Format) will be executed and the client will be logged out by the backend system.

Please note the difference between application-level heartbeat broadcasts sent by the backend system and AMQP-level heartbeats:

- AMQP heartbeats are defined by the AMQP protocol specification and allow the client to monitor the existence of a connection between the client and the AMQP server.
- Heartbeat broadcasts published by the backend system are proprietary to the Public Message Interface and allow the client to monitor the availability of the backend system.

3.2.2 Market Data Broadcasts

A durable topic exchange per client is setup on the AMQP Server.

- `m7.broadcastExchange.<login-id>`

The backend system sends messages to the specific trader broadcast exchange whenever a respective action has been performed. Those messages will be delivered to the traders private queue that will also be setup in the AMQP Server. The private clients queue will be named

- `m7.broadcastQueue.<login-id>`

The backend system sends broadcasts messages whenever a change occurred either initiated by any traders or the backend system itself. Based on the current data model used in the backend system different routing keys are used to delivery broadcast messages to the traders private exchanges and queue. See Message Format section to find out what routing key is used by the backend system upon a market data change.

A client can subscribe to broadcasts by binding a consumer to its private queue. Clients cannot subscribe to all existing queues. The access is controlled by configuring privileges for broadcast queues based on the trader's assignments. If access to a broadcast queue is not granted, the client will not be able to bind a consumer to its private queue.

Example A message (formatted according to schema 6.0) containing information about an order entry done by *TM001-BG1------X* for product *Intraday_Power_D* and delivery area *10YDE-RWENET---I* will be sent using the routing key:

```
6_0.Intraday_Power_D.10YDE-RWENET---I.TM001-BG1------X
```

The message will be pushed to all private queues bound to exchanges using the same routing key.

3.2.3 Reference Data Broadcasts

As each trader has its private broadcast queue also reference data messages will be send via the private queue using specific routing keys. Only messages send by the backend system that match the routing key will be delivered to the trader's private queue.

The backend system sends messages to the traders broadcast queue whenever it needs to publish a reference data change to the clients.

A client can subscribe to broadcasts by binding a consumer to its private queue.

Not all traders can receive all broadcasts. For each trader, the access to its private queue is controlled by configuring privileges. If access to a broadcast queue is not granted, the client will not be able to bind a consumer to its private queue.

Example A message (v6.0 format) containing information a suspension of trader CXDBSX01 will be sent using the routing key:

```
6_0.CXDBSX01
```

The message will be pushed to the trader's queue that is bound to the exchange using the same routing key:

```
m7.broadcastQueue.CXDBSX01
```

3.2.4 Sequence counting for Broadcast Messages

There will be a sequence number used to identify the order of the broadcasts and to find out if some broadcasts have been lost. The sequence number is not part of the payload but it is stored within the header of the AMQP message as an attribute called `x-m7-group-sequence`.

The sequence will be always increased by one for the next broadcast. It will be in-memory only (NOT persistent) which means that when the backend system shuts down or terminates, the sequence will be reset to 0. Whenever the client gets a value which is not expected (i.e. value different than last_value+1) it should request the market data from the backend system.

The sequence number is counted based on the routing keys (attribute `x-m7-group-id` in message header). So for each routing key there will be a different sequence number. All queues bound to the default broadcast exchange with the same routing key will receive the same sequence id.

Please note, the routing keys will be respecified in final version of this document, such that one broadcast routing key is used only for one message type.

3.2.5 How to Receive Broadcasts

As the traders private broadcast queues are durable queues and already setup in the AMQP Server any client that wants to subscribe to broadcasts simply registers a consumer for its private queue.

Whenever a message is broadcasted by the backend system, a copy of the message will be placed into the trader's private queue and the clients call back method will be invoked on the registered consumer.

To stop receiving broadcasts, the client should remove the consumer from its private queue.

The broadcast queues within the AMQP Server are setup as durable queues to avoid any loss of messages due to any connection problems. The broadcast queues are also configured with a time to live for messages put into the queue (e.g. 60 seconds). This will protect the AMQP Server from out of memory in case a client consumes the messages to slow or any other connection problems.

3.2.6 Acknowledgement of Broadcasts

Client must not leave any unacknowledged messages on AMQP server. Client applications are requested to use auto acknowledgement on RabbitMQ channels to acknowledge the reception of all response messages as soon as the message has been received (before processing of the message). It is strongly recommended to implement gap-detection mechanism as described in chapter 3.2.4 Sequence counting for Broadcast Messages to prevent data-loss.

If server side queue gets filled with unacknowledged messages, it might lead to high memory consumption. The private broadcast queues on server side are constantly monitored and in case one of the queues hit a certain threshold, the connection might be actively disconnected by the server and the Application ID gets deactivated.Failover Processing

In case of AMQP server shutdown (due to failure or restart), the client subscriptions are lost. If the client has registered a shutdown listener, it will receive a shutdown notification from AMQP. After successful reconnect to the AMQP server, the clients have to re-subscribe.

In case of backend system failure, the client subscriptions will stay active, but clients will not receive any broadcasts until the backend system is restarted. Once the backend system is restarted, delivery of broadcasts will resume with no further actions

required on the client side. Client will recognize that the backend system is down because no heartbeat broadcasts are sent anymore. Note that in the case of a backend restart, the sequence numbering for the requests will be restarted.

Any broadcast messages published by the backend system while a client was disconnected will be lost for that client if the client does not reconnect within the time specified as time to live for the messages within the queue.

3.2.7 Flow Control

When a client is consuming messages too slowly, a backlog of messages waiting for processing may build up in the private queue(s) owned by this trader.

If messages exceed the time to live limit, the AMQP server will start deleting those messages from the trader's private queues. Thus, a slow client may lose messages.

The clients are therefore required to consume and acknowledge each message as soon as possible. If the processing of a message is a non-trivial task, the receiver must handle the receiving of the message separately from the actual processing of the message: the message must be consumed, acknowledged and stored in a memory buffer of the client, where it awaits processing.

It is important that clients consume messages as quickly as possible. Should a client fail to adhere to these rules due to slow consuming, it may lose messages and display a non-up to date market state.

3.2.8 Message type

M7 system uses AMQP message attribute named `type` to provide information about the content of each message. Example: `ContractInfoRprt`.

This attribute can be used by client application to determine content of each message even before unpacking/processing.

4 Security

All communication between the client and the AMQP server is encrypted using the Transport layer security (**TLS) version1.0**. Client and server certificates are used to establish a trusted connection. The usage of asymmetric encryption ensures confidentiality, authentication, message integrity assurance and non-repudiation of origin.

4.1 Server Certificate

The backend system uses a Server Certificate signed by a well-known and trusted *Certificate Authority* (CA) like VeriSign or Comodo.

Usually the CA root certificates from these CAs are known and trusted by the software development frameworks like Java or .NET. If not, clients need to add the CA root certificate to a list of trusted certificates. The needed CA root certificate files can be downloaded directly from the CAs Website[3].

The exact way how this is done depends on the platform and programming language in which the client application is developed. Please note that the CA root certificate has to be imported into a location used by the client application's runtime environment, not into a web browser.

4.1.1 Using CA Root Certificate in Java

For Java clients, the CA root certificate must be imported into a *keystore*, which will be used to initialize the *Trust Manager* used by the client application. The certificate can be imported using the `keytool` utility, which is part of the Java runtime environment:

```
keytool -import -alias m7-ca -file <cacert> -keystore <keystore>
```

In the command above, `<cacert>` is path to the CA certificate file in PEM format (`cacert.pem`) and `<keystore>` is path to the keystore file. You have to confirm that you trust the certificate being imported.

The keystore file will be created if it does not exist yet. Java keystores are protected by password; choose a password when first creating the keystore and then use it whenever accessing the keystore.

Refer to Java documentation for more information about the `keytool` utility. Example code can be found in the reference client implementation.

4.2 Client Certificate

An organization that wants to use the Public Message Interface of the backend system has to apply for an account on the AMQP server. The following is provided when a new account is set up:

- AMQP server host names, port number and virtual host name.
- Trader's login ID if not already existing.
- Client certificate and private key that will be used by the client when connecting to the AMQP server.
- CA root certificate that has to be imported as a trusted certificate on the client side.

The client certificate:

- complies to X.509v3 specification,
- uses key length of 2048 bits,
- subject contains an unique identifier as the *Common Name*,
- is signed by Deutsche Börse CA,
- is provided in format PKCS#12 (`.p12`).

[3] VeriSign: http://www.verisign.com/support/roots.html
Comodo: https://support.comodo.com/index.php?_m=downloads&_a=view&parentcategoryid=1

The client's private key is used to verify the client's identity when communicating with the AMQP server. Should a third party get hold of the client's private key, it could forge client's identity and access the encrypted communication with the server.

It is therefore extremely important to keep the private key secure.

4.2.1 Using Client Certificate in Java

To be able to use the client certificate and private key in a Java client, they need to be loaded into a keystore, which is used to initialize a *Key Manager*. Java supports the PKCS#12 format (`.p12`), where the private key is protected by a passphrase. The passphrase is provided to the client along and it must be used when loading the `.p12` file into the keystore.

Please refer to the reference client implementation for sample Java code. For further details regarding SSL security, please consult the SSL specification, AMQP specification and www.rabbitmq.com/ssl.html.

4.3 Authentication

When a client connects to the AMQP server using the SSL protocol, both peers are mutually authenticated by exchanging their certificates during SSL handshake. All subsequent communication between the client and the server is encrypted using a key and encryption algorithm agreed on during the handshake. For connecting to the AMQP Server the username and password is required. The AMQP Server is connected to an internal LDAP Server which will verify the given credentials. In case of an unsuccessful authentication the client won't be able to connect to the AMQP Server.

Whenever a client sends a request to the AMQP server, it uses a client-specific exchange. The exchange name contains the client login id, which can be extracted by the backend system when processing the request to identify from which client the request came.

4.4 Authorization

Authorization of client actions occurs on two levels:

4.4.1 Authorization by AMQP server

The AMQP server verifies client's privileges when a client accesses resources stored on the AMQP server. This includes exchanges and queues that the client tries to configure, read or write. This applies to:

- binding queues to private broadcast exchanges
- sending messages to request exchanges

In case of insufficient privileges, the attempt to access the resource is immediately (synchronously) rejected by AMQP.

4.4.2 Authorization by the backend system

The backend system verifies privileges when processing requests from clients.

In case of insufficient privileges, the request is rejected by the backend system. From the client point of view, this rejection occurs synchronously for inquiry requests as a negative response message is sent to the traders response queue and asynchronously for management requests as a negative response message is sent to the traders private exchange using the routing key `<schema-version.login-id>`.

5 Message Format

All messages sent between the backend system and clients have XML-encoded payload and specific AMQP message properties. This section describes the xml payload format and the AMQP message properties in detail. The xml payload format specification comes in the form of XML schemas, which specify the allowed structure and format of XML elements and attributes.

5.1 General Information

5.1.1 AMQP Message Properties

Following message properties have to be set when sending a request to the backend system:

AMQP Message Property	Description
content-type	Contains information about the used XML payload version as well as the used message type. Valid content-type definitions are (version number has to be filled with the used version): ▪ x-m7/request; version=x (Used by the clients when sending requests) ▪ x-m7/response; version=x ▪ x-m7/broadcast; version=x ▪ x-m7/heartbeat; version=x ▪ x-m7/error; version=x
reply-to	contains the predefined trader's queue name a response has to be sent to (See 3.1)
user-id	contains the login-id of the logged in trader
app-id	contains the application id given by the granting authority
correlation-id	contains the request message id generated by each client
expiration	contains an optional entry specifying if the request should be deleted if not executed within the specified time
contentEncoding	contains ***gzip***, if messages are compressed (content is encrypted using gzip method); property is null if messages are not compressed
header	can contain connecting application version in format (optional) **app-version**:version where version must be in integer(s) format optionally separated by dots (*ie. 10 or 4.1.5*)

If app-version is provided M7 checks if it's higher or equal than minimal version for the given app-id. If app-version is lower, the connection is refused.

If app-version is not provided, the check is not performed and connection is allowed.

5.1.2 XML Convention

- Element tags are only used for structure reasons.
- Data is only carried in attributes, never in element tags.
- Types:
  - All Elements are bold, Attributes are non-bold
  - **SE:** Structure Element. No Data embedded between tags, but may contain attributes. Contains no data. (grey background and bold)
  - **CE:** Content Element. Data is embedded between tags, can also contain attributes.(bold)
  - **A:** Attribute of an Element.
- The sorting of elements and attributes is not guaranteed and might change in the future.
- **m/o** column specifies if presence of element/attribute is mandatory or optional

- value "**c**" means conditional. Such attribute is marked as optional in API, but may be required from the service depending on system settings or particular state of reference data. Such condition is always specified.

5.1.3 Standard Message Header

Every message will contain a header at the beginning of the message.

XML Tag	Type	m/o	No.	Data Type	Short description
StandardHeader	SE	m		Structure	
marketID	A	m		Char(4)	Market Identification Code (MIC) of the market to which the request is sent or from which the request originates.
onBehalfUserId	A	o		Integer	UserID of the target user in case of On-behalf trading (see 5.1.10).
clientData	SE	o	0..1	Structure	
clientDataInt	A	o		Integer	For order management transactions the client data fields in this section can be used by the client to store information or meta-data about a request. None of this information is used by the backend and it is returned to the client with the response.
clientDataString	A	o		String	
clientCorrelationId	A	o		String	

Table 1: Message layout of the Standard Message Header.

5.1.4 XML Validity and Data Binding

It is important that clients produce valid XML code when sending requests to the backend system. The backend system uses a validating parser, which rejects any requests that do not strictly conform to supported XML schemas.

To make parsing and creating of XML data easier, and to avoid problems with validity of the XML code, clients are encouraged to use XML data binding.

5.1.5 Schema Version

Every message sent or received by the backend system must specify in its metadata the XML schema version that was used to encode its payload. This information is stored in message properties in field contentType as a string.

This information can be used to validate that both peers use the same version of the payload format. The backend system rejects requests with an unexpected schema version.

5.1.6 Heartbeat

The heartbeat contains the message SYSTEM_ALIVE:<interval> and the message attribute "server-timestamp" within the header property of each message.

5.1.7 Quantity values in Messages

Quantity values in all messages (requests and responses) are given as integer values which represents exactly the database values stored in the backend. The interpretation/display of these values depends on the product attributes *decShftQty*, minQty and *qtyUnit* (see 6.4.19).

The attribute *decShftQty* defines the position of the decimal point within the integer value (e.g. the integer value 1000 with a *decShftQty* of 3 means a decimal number of 1.000).

The attribute *minQty* defines the possible quantity steps which can be entered into the system and at the same time the smallest possible quantity value (e.g. a *minQty* of 100 means, quantities can be entered in 100 steps starting with 100: 100, 200, 300, etc. – the value of 50 would be rejected). The *minQty* can also be used to limit the number of displayed decimal places for a quantity value: With a *minQty* of 100, the last two zeros might be cropped when displaying quantities, because they are always zero.

The attribute *qtyUnit* contains a string with the unit of the quantity values (e.g. "MW" for megawatt in the power market).

Table 2 shows some examples.

decShftQty	smallestTrdUnit	qtyUnit	Value in messages	Possible display value
3	100	MW	1300	1.3 MW
3	100	MW	700	0.7 MW
3	1000	MW	34000	34 MW
0	100	Ltr.	700	700 Ltr.

Table 2: Examples for displaying quantity values

5.1.8 Price values in Messages

Price values in all messages (requests and responses) are given as long values which represents exactly the database values stored in the backend. The interpretation/display of these values depends on the product attributes *decShftPx* and *currency* (see 6.4.19).

The attribute *decShftPx* defines the position of the decimal point within the long value (e.g. the long value 1276 with a *decShftPx* of 2 means a decimal number of 12.76).

The attribute *currency* contains a 3 character long identifier for the currency. A *decShftPx* of 2 for the currency *Euro* means that the values are given in Eurocents (3499 = 34.99 EUR = 3499 Eurocents).

5.1.9 Date time values in Messages

Date time values in messages (data type in the message layout tables is "DateTime") are given as XML Schema DateTime values in the following format:

`YYYY-MM-DD`**`T`**`hh:mm:ss.sss`**`Z`** `(2012-09-14T12:34:23.351Z)`

Symbol	Description	Example
YYYY	The year	2012
MM	The month	09
DD	The day of the month	14
T	Separator between the date and the time section	T
hh	The hour of the day (24 h)	12
mm	The minute of the hour	34
ss	The seconds of the minute	23
sss	The milliseconds of a second	351
Z	Time zone - Zulu time zone = UTC time	Z

All Date/Time values are given in UTC time zone. To get local times, the client have to add or remove hours based on the local time zone.

Contract naming patterns are affected by Timezone set on product level.

Market timezone can be obtained using SystemInfoResp message (see 6.1.5)

5.1.10 On-behalf Trading

The backend system supports trading on-behalf of another user. There are three different levels of on-behalf trading:

- *On-behalf trading on member level*: Traders having the right to trade on-behalf are allowed to perform actions for all traders belonging to the same member, same balancing group and product configuration.
- *On-behalf trading on broker level*: Broker users are allowed to trade on-behalf for all traders assigned as the assigned members for their account regarding the user product configuration.
- *On-behalf trading on admin level*: All Admin users are allowed to trade on-behalf for all traders in the system regarding the user product configuration (only products assigned to trader can be traded on behalf even if admin himself has wider products assignments).

In order to submit requests to the backend as on-behalf requests, the standard message header field *onBehalfUserId* must be filled with the user ID of the target user, for whom the request is sent on-behalf (see 5.1.3). Although the standard message

header is part of every message in the Public Message Interface, the field *onBehalfUserId* is only relevant for the following request types:

- Order Entry Request (OrdrEntry)
- Order Modify Request (OrdrModify)
- Order Request (OrdrReq)
- Order Execution Report (OrdrExeRprt)
- Pre-Arranged Order Processing (PreArrangedOrdrProcess)
- Trade Recall Request (TradeRecallReq)
- Message Request (MsgReq)
- Message Report (MsgRprt)
- Trade Capture Request (TradeCaptureReq)
- Trade Capture Report (TradeCaptureRprt)
- Delete Quotes Request (DeleteQuotesReq)
- Order Limit Request (OrdrLmtReq)

If the field is filled for a request, which does not support on-behalf trading, it will be ignored by the backend.

In case of an on-behalf request, the user with the *onBehalfUserId* will be retrieved on backend side and the user context of the request is changed to this user. This means, the request is treated by the backend like a normal request directly from the user.

5.1.11 Message properties summary table

The description for every message starts with a table that summarizes key properties of the message. The following table describes the different properties and their meaning:

Message classname is present in the table header.

Property	Description
Type	Type of the message: • **Inquiry Request**: Message to retrieve information out of the backend system during the initial load phase or in case of a detected gap. • **Inquiry Response**: Response message to an Inquiry Request. • **Management Request**: Message to send new business information to the backend system for changing business objects. • **Management Response**: Response message to a Management Request. • **Broadcast**: Message is send as broadcast initiated by the backend system. One message can have several types.
Roles	User roles that are allowed to send or receive the message. Possible values: Trader, Market Operations, Market Makers, Brokers, Sales, Data Vendor, Settlement Operations and <ALL>
Routing Keys	Routing key(s) used to decide message queue destination. (request messages only)
Response To	Request message to which the response is a reply message (response messages only). If not specified, the message is a broadcast-only message.
Broadcast	Indicates if the response message can be sent as a broadcast (response messages only). If 'NO' then the message is only sent as a reply to a request.
Broadcast Routing Keys	Routing keys specified for a broadcast message. Empty for non-broadcast responses (response messages only) Broadcast routing keys will be deprecated and may be decommissioned in next releases. For documentation purpose they will be replaced by "Broadcast audience" information.
Broadcast Audience	Contains an audience scope for Broadcast message.
Request Limits	Request limit values for Inquiry Request (see also 3.1.9.1): <shortTermLimit>/<longTermLimit> (Example: 1/10; max 1 request every minute and 10 requests per hour) The limits given in this document are the default values. They can be changed during runtime of the backend system, if necessary.

5.1.12 Any elements and attributes

For forward compatibility reasons each entity in PMI is now expandable with “any” element and “anyAttribute” in XSD schema. For more details please refer to chapter 10.

6 Public Requests and Responses

The requests and responses described in this chapter are used for users without administrative privileges to communicate with the backend.

6.1 General Requests and Responses

The general requests and responses described in this chapter are used for login and logout to the backend system as well as for responding to management requests.

6.1.1 Login Request (LoginReq)

LoginReq	
Type:	Inquiry Request
Roles:	<All>
Routing Keys:	`m7.request.inquiry`
Request Limits:	3/20

Example:

XML Tag	Type	m/o	No.	Data Type	Short description
LoginReq	SE			Structure	
StandardHeader	*SE*	*m*		*Structure*	*Standard header of each message. Please see 5.1.3.*
user	A	m		String	Login ID of the user that want to login to the backend system.
force	A	m		Boolean	Flag that indicates if this user want to force a login even if a user with the same credentials is already logged in into the backend system.
disconnectAction	A	m		String	Action that will be executed in case of an unexpected connection loss. The following values are allowed: "**NO**": No action is executed. "**DEACT_USER_ORDRS**": All orders of this user will be deactivated. "**DEACT_ACCT_ORDRS**": All orders entered into the backend system of the assigned trader accounts will be deactivated.

The Login Request is sent to the backend system to participate in the market. As a response to this request either a User Response (successful login – see 6.6.1) or an Error Response is returned. The Error Response might indicate that a trader with same credentials is already logged in into the backend system.

Table 3: Message layout of a Login Request.

6.1.2 Logout Request (LogoutReq)

LogoutReq	
Type:	Inquiry Request
Roles:	<All>
Routing Keys:	`m7.request.inquiry`
Request Limits:	3/20

The Logout Request has to be sent by a client application if the trader logs out of the backend system manually.

Table 4: Message layout of a Logout Request.

6.1.3 Logout Report (LogoutRprt)

LogoutRprt	
Type:	Inquiry Response, Broadcast
Response to:	LogoutReq (sent to private response queue see 3.1)
Broadcast:	Yes
Broadcast Routing Keys:	`<schema-version>.trdr.<login-id>`
Broadcast audience	`Only user with usrId (in case of the concurrent login).`

Roles:	<All>

The Logout Report is used in 2 cases:

- as a response to the Logout Request (to the private response queue see 3.1). In this case the attributes usrId and sessionId will have the value "0".
- as a broadcast to the session of the user who is logged out as a consequence of concurrent login with the same user credentials and the 'force' attribute set to "true" (routing key: `<schema-version.login-id>`) In this case the usrId will be set to the usrId of the user and the sessionId will be set to the sessionId that has been forced to log out.

The following table shows the format of the Logout Report:

Table 5: Message layout of a Logout Report.

6.1.4 System Info Request (SystemInfoReq)

SystemInfoReq	
Type:	Inquiry Request
Roles:	<All>
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

The System Info Request is used to get general information about the backend system (see response message description below).

Table 6: Message layout of a System Info Request.

6.1.5 System Info Response (SystemInfoResp)

SystemInfoResp	
Type:	Inquiry Response
Response to:	SystemInfoReq (sent to private response queue see 3.1)
Broadcast:	No
Broadcast Routing Keys:	---
Roles:	<All>

The System Info Response returns general information about the backend system as reply to a System Info Request.

Table 7: Message layout of a System Info Response.

6.1.6 Acknowledgement Response (AckResp)

AckResp	
Type:	Management Response
Response to:	OrdrEntry; OrderModify; PreArrangedOrdrProcess; ModifyAllOrdrs: TradeRecallReq: (sent to private response queue see 3.1)
Broadcast:	No
Routing Keys:	---
Roles:	Trader, Market Operation

The Acknowledgement Response is sent upon reception of any management request. If an acknowledgement response is not sent, the client has to send an inquiry request to figure out if the previously sent management request has been executed.

The response is sent back using the same correlation id within its message attributes. Acknowledgement Responses are always sent to the private response queue of the client.

Note: because of the new architecture constraints, the clOrdrId was removed from this message.

Table 8: Message layout of an Acknowledgement Response.

6.1.7 Error Response (ErrResp)

ErrResp

Type:	Inquiry Response; Management Response; Broadcast
Response to:	<All> (sent to private response queue see 3.1
Broadcast:	Yes
Broadcast Routing Keys:	`<schema-version>.trdr.<login-id>`
Broadcast audience:	`Connected user whose action caused asynchronous error (ie. OrdrEntry while not having sufficient cash limit).`
Roles:	<ALL>

The Error Response is sent whenever the backend system determines a business exception based on wrong entries in the request.

Error Responses can be sent synchronously or asynchronously (broadcast). The backend system sends the Error Response in a synchronous way whenever the received xml contains syntactical errors. In this case the requests have not been processed by the backend system yet. Error Responses are sent by the backend system in an asynchronous way if any errors happen during processing of the request. Synchronous Error Responses get sent to the private response queue of each trader, asynchronous get send to the private traders broadcast queue.

Any connected application may use English text provided in "err" attribute or can use provided (as part of the reference implementation) text resource to fill with the transferred variables . This way the localisation of error text can be done on client side (server side provides no localisation in the err texts).

Example *Message contains:*

```
<Error errCode="12345" err="ACCT1 - account not found">
  <VarList>
    <Var id="0" value="ACCT1"/>
  </VarList>
</Error>
```

Provided resource file contains:

errCode	*Error string English*
12345	*{0} – account not found*

Client application can either use err attribute from message which will always be in English or use provided resource file to translate to any language and fill provided variable itself.

Table 9: Message layout of an Error Response.

6.1.8 Change password request (ChgPwdReq)

ChgPwdReq	
Type:	Management Request
Roles:	<All>
Routing Keys:	`m7.request.management`
Request Limits:	1/10

This message can be used by any user to change his password. It is not possible to change password on behalf.

As a result one of following actions is performed

1) ErrResp: The change of password was not successful. Attribute err will contain detailed error message from LDAP/M7.
2) If change was successful shutdown signal from broker with reason PWD_CHANGE is sent and user then can relogin with new password.

Table 10: Message layout of a Change Password Request

6.2 Order Entry and Maintenance

The messages described in this section are used to enter and maintain orders. These messages can be used only by trading participants and are not available for Market Operation users.

6.2.1 Order Entry (OrdrEntry)

OrdrEntry	
Type:	Management Request
Roles:	Trader, Market Operation
Routing Keys:	`m7.request.management`

The Order Entry message enables a trader to send up to 100 orders at once to the backend for execution. The orders are contained in a so called basket. It is possible to define the execution restriction on a basket level.

An order entry triggers the following message flow:

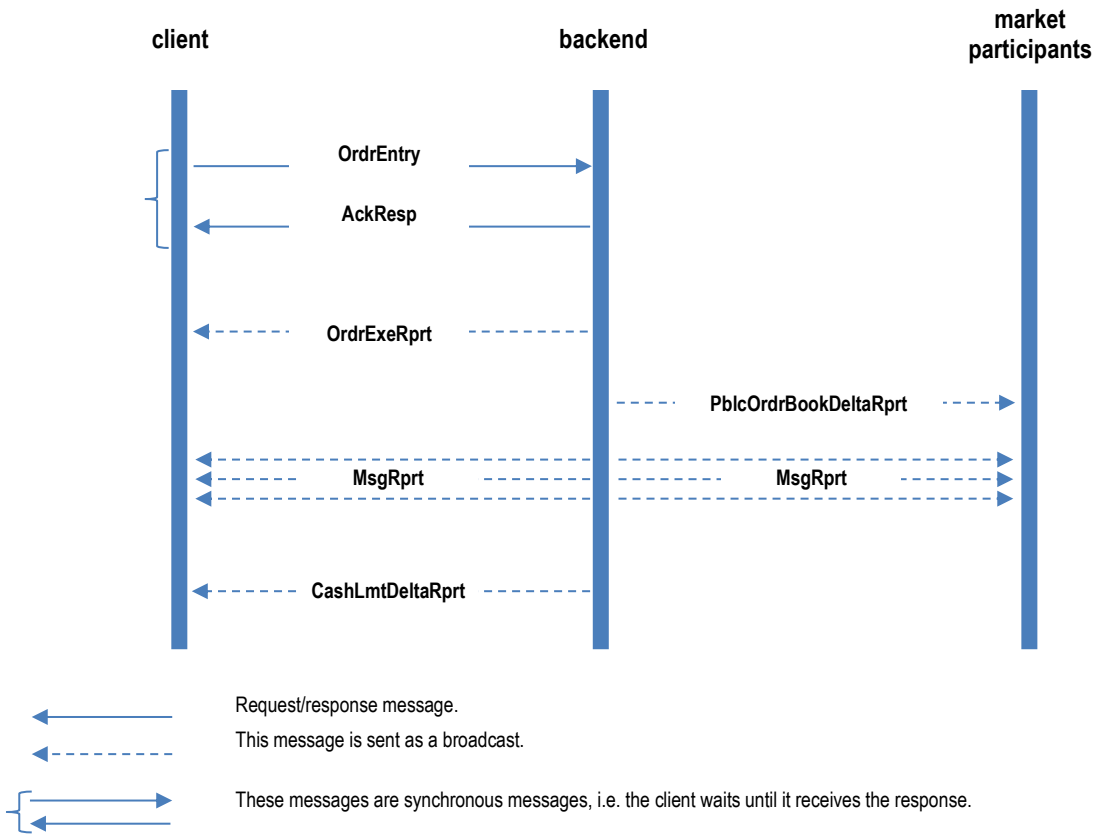


Figure 1: Message flow triggered by an Order Entry.
Actions caused by partial or full execution or invalid order parameters are not part of this diagram.

Only one acknowledgement (AckResp) or one Order Execution Report (OrdrExeRprt) is sent back, even if the OrdrEntry request contains several orders.

In case of order executions (trades) public and private messages will be broadcasted (MsgResp). In addition to the OrdrExeRprt a Trade Capture Report will be sent to the affected parties, the public order books will be updated (giving rise to Public Order Book Delta Reports) and the connected data vendors will receive also a data update.

A Cash Limit Delta Report is sent only in case the order entry has an impact on the cash limit. If the order entry has no impact on the trading cash limit, this message will not be sent.

Q order type can not be mixed with other order types in one message (otherwise ErrResp is returned)

W order type can not be mixed with other order types in one message (otherwise ErrResp is returned)

The message layout is defined in the following table.

Table 11: Message layout of an Order Entry message.

6.2.2 Order Modify (OrdrModify)

OrdrModify	
Type:	Management Request
Roles:	Trader, Market Operation
Routing Keys:	`m7.request.management`

This message is used to modify one or several orders. If the order modification has impact on the order execution priority, the old order will be removed by the system and a new order with a new order id will be entered instead. As a result, after an order modification with impact on the execution priority, two OrdrExeRprt messages are returned:

- First the deletion of the modified order is reported.
- Secondly, an Execution Report indicating the newly added order is returned.

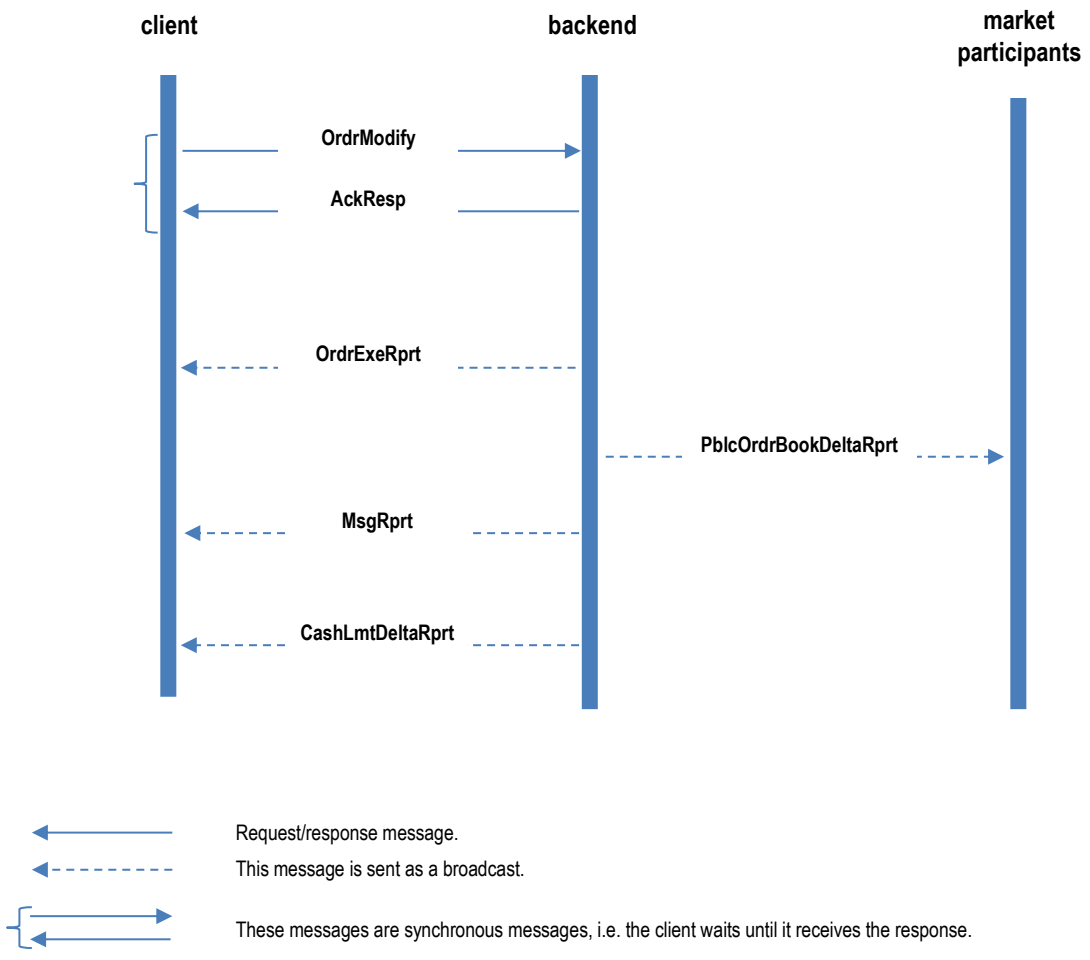


Figure 2: Message flow triggered by an Order Modification.

The message layout of the order modification message (OrdrModify) is described in the following table.

Table 12: Message layout of an Order Modify message.

6.2.3 Order Request (OrdrReq)

OrdrReq	
Type:	Inquiry Request
Roles:	Trader, Market Operation
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

The Order Request is used to request all own orders, which are currently active or hibernated for the given account and products. The Order Request is answered with an Order Execution Report (see 6.2.4).

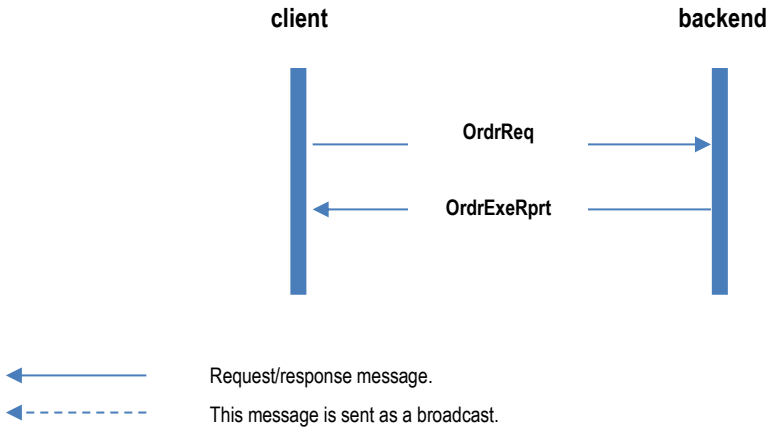


Figure 3 - Message flow triggered by an Order Request.

The following table lists the message layout for the Order Request

Table 13: Message layout of an Order Request.

6.2.4 Order Execution Report (OrdrExeRprt)

OrdrExeRprt	
Type:	Management Response; Broadcast
Response to:	OrdrEntry; OrdrModify; OrdrReq; ModifyAllOrdrs; (sent to private response queue see 3.1)
Broadcast:	Yes
Broadcast Routing Keys:	`<schema-version>.bg.<acctId>`

Broadcast audience:	`Trader (owner of the order) and traders from his Balancing groups.`
	`Admins`
	`Brokers with assignment to Trader(owner of the order).`

Roles:	`Trader, Market Operation`

The Order Execution Report is sent in the following cases:

- After a successful Order Entry.
- After an Order Modify request.
- After a partial or full execution of the order.
- In case of the execution of a Pre-arranged order.

As a response to an Order Request, the Order Execution Report is sent to the private response queue of the requesting user see 3.1.

In both cases, traders will receive only the orders entered in the account they are assigned to, whereas market operations users will receive all orders

Table 14: Message layout of an Order Execution Report.

6.2.5 Pre-Arranged Order Processing (PreArrangedOrdrProcess)

PreArrangedOrdrProcess	
Type:	Management Request

Roles:	Trader, Market Operation
Routing Keys:	`m7.request.management`

The pre-arranged order processing request is used to accept or reject a pre-arranged trade.

The message flow generated by this request is depicted in the following graph.

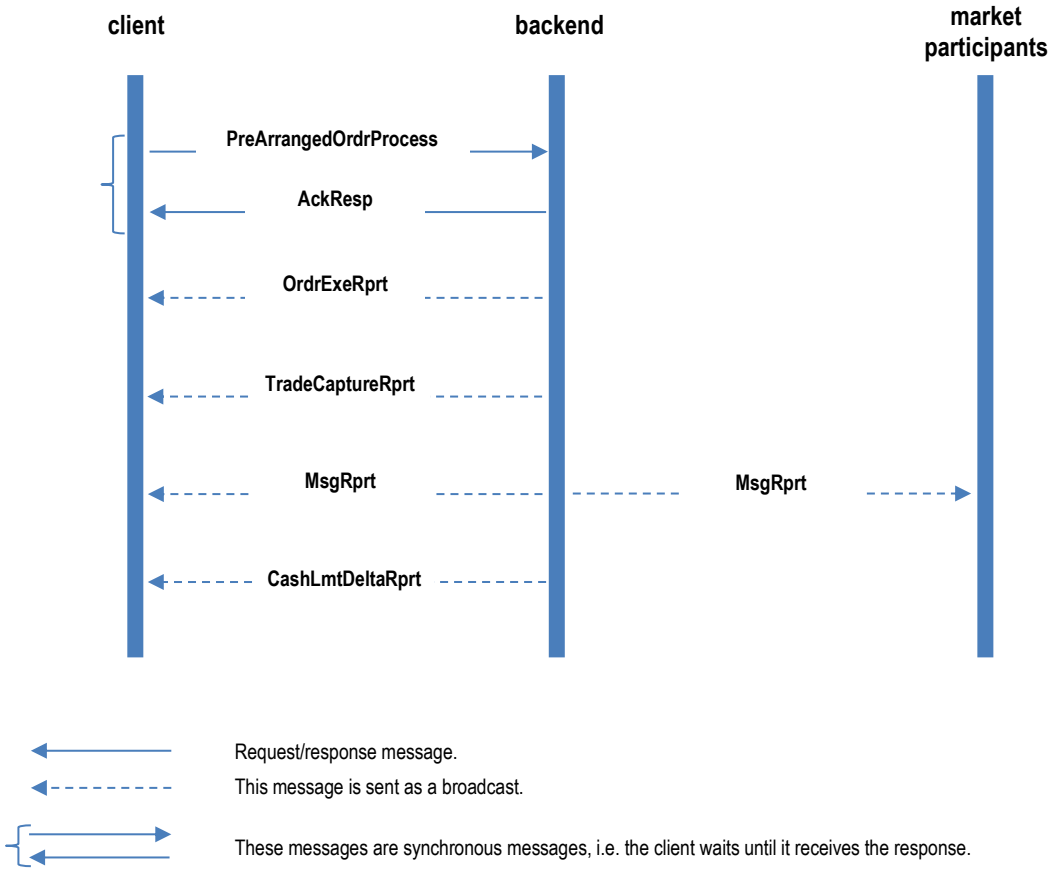


Figure 4: Message flow initiated by a pre-arranged order processing request.

Table 15: Message layout of a Pre-Arranged Order Processing message.

6.2.6 Modify All Orders (ModifyAllOrdrs)

ModifyAllOrdrs	
Type:	Management Request
Roles:	Trader, Market Operation
Routing Keys:	`m7.request.management`

The Modify All Orders message is used to delete or deactivate all orders belonging to a member, an account or a trader. Only one of those attributes should be filled with a proper value. In case of an account also specific products can be defined to be taken into account.

The message flow generated by this request is depicted in the following diagram:

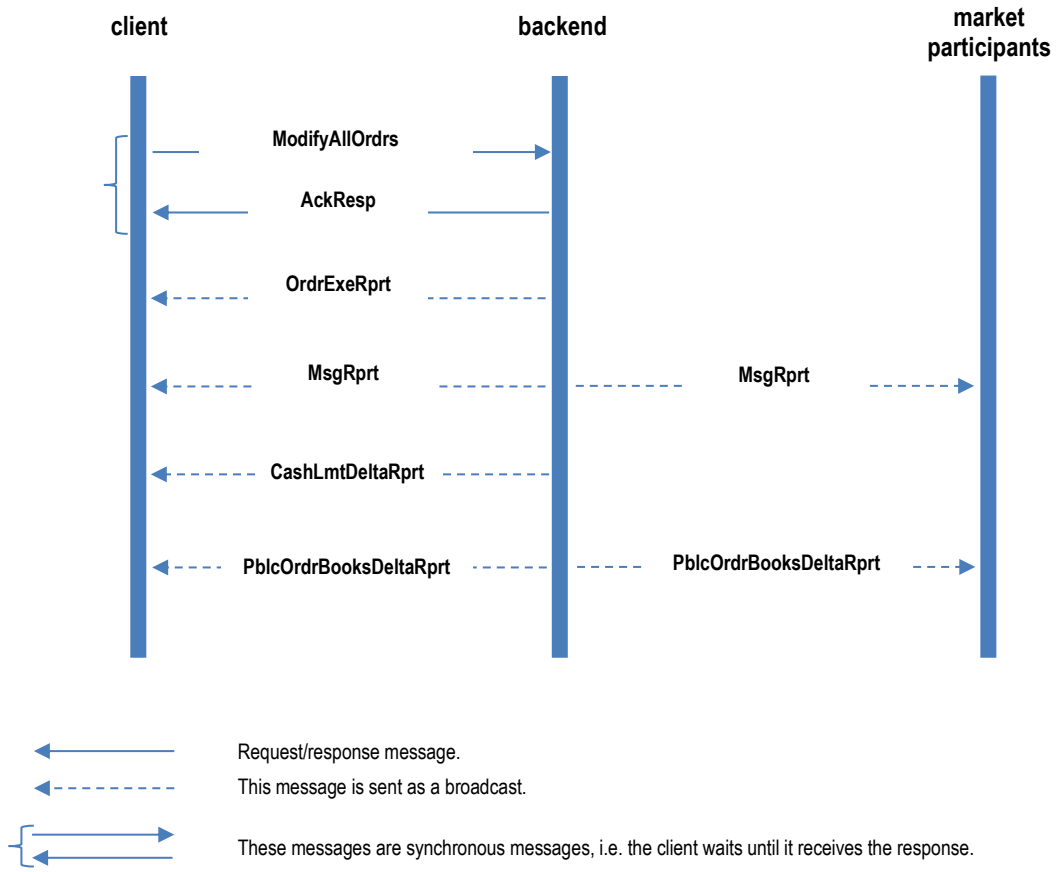


Table 16: Message layout of a Modify All Orders message.

6.2.7 Order Limit Request (OrdrLmtReq)

OrdrLmtReq	
Type:	Inquiry Request
Roles:	Trader, Market Operation, Broker, Market Maker
Routing Keys:	`m7.request.inquiry`

The OrdrLmtReq is used to retrieve the current Order Limits valid for the member to which the logged in trader is assigned.

Table 17: Message layout of Order Limit Request message

6.2.8 Order Limit Report (OrdrLmtRprt)

OrdrLmtRprt	
Type:	Inquiry Response; Broadcast
Response to:	OrdrLmtReq (sent to private response queue see 3.1)
Broadcast:	Yes
Broadcast Routing Keys:	`<schema-version>.mbr.<mbrId>`
Broadcast audience	`All traders from particular member` `Admins`
Roles:	`Trader, Market Operation, Broker, Market Maker`

The OrdrLmtResp is returned as a response to the OrdrLmtReq or as a broadcast as a result of an event that changes an Order Limit. The message lists all the Order entry limits of the inquired member.

These limits are then checked at every order entry process for appropriate member.

Table 18: Message layout of Order Limit Response message

6.3 Trade Maintenance

6.3.1 Trade Recall Request (TradeRecallReq)

ModifyAllOrdrs	
Type:	Management Request
Roles:	Trader, Market Operation
Routing Keys:	`m7.request.management`

This message is used to request a recall of a miss-trade. Market Operation will be informed about the trade recall request after successful submission. The message flow is shown in the chart below.

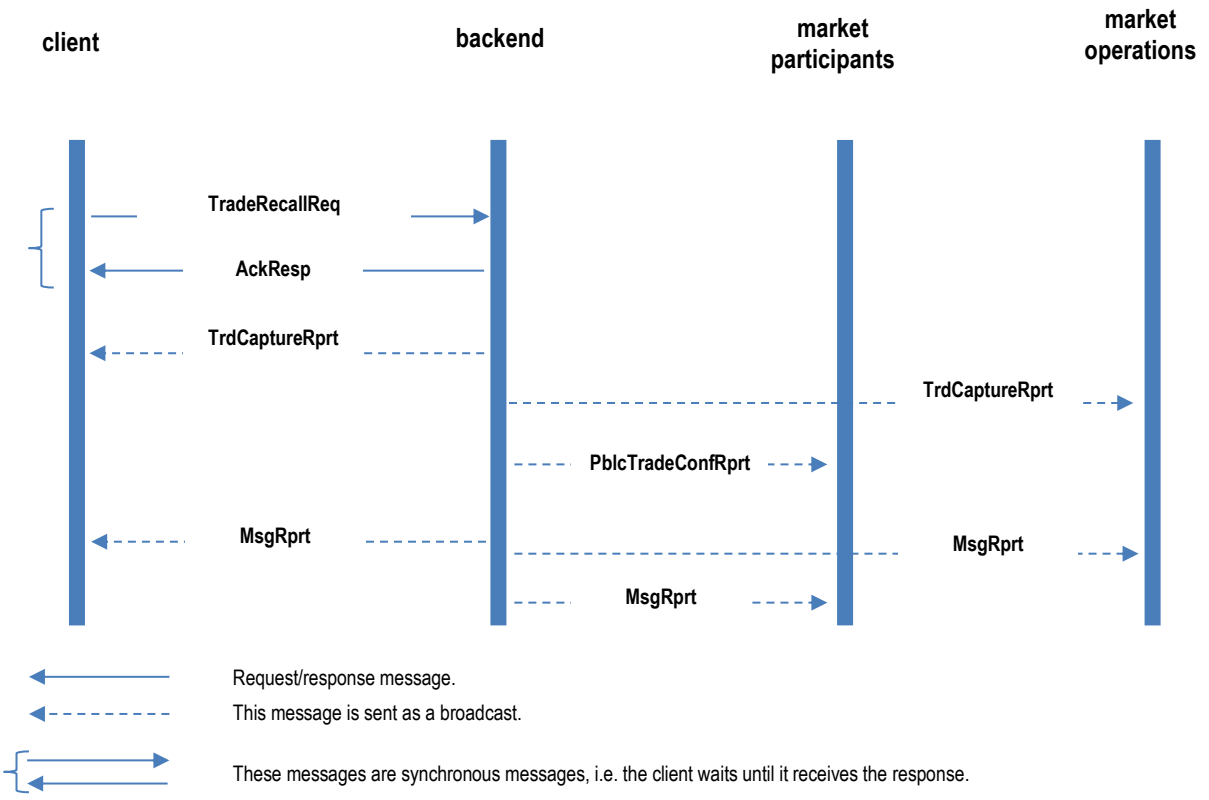


Figure 5: Message flow triggered by a Trade Recall Request.

The message required for a trade recall request contains only the trade identifier and revision number of the affected trade.

Note: if trade for which recall is requested has filled parentTradeId, the trade with tradeId=parentTradeId and also all other trades with the same parentTradeId will be recalled simultaneously.

Table 19: Message layout of a Trade Recall Request.

6.3.2 Prearranged Trade Entry (PrearrangedTradeEntry)

PrearrangedTradeEntry	
Type:	Management Request
Roles:	Market Operation, Broker
Routing Keys:	`m7.request.management`

The message is used by broker to enter the prearranged trade. It is used for On Exchange prearranged trades and private and confidential trades.

Order Execution Reports and Trade Capture Reports are generated as a result of the trade.

Table 20: Message layout of Prearranged Trade Entry.

6.4 Market Information

6.4.1 Retrieval of Public Order Book information

The public order book of a contract is build up by performing the following steps:

- Retrieve the initial data set at the start of a user session, using the Public Order Books Request (PblcOrdrBooksReq). All active orders in the order book at request time are returned.
- During the session, process the Public Order Books Delta Responses (PblcOrdrBooksDeltaRprt) that contain all updates to the order books.

PblcOrdrBooksDeltaRprt messages with order book revision numbers smaller than the ones received in the PblcOrdrBooksResp must be ignored!

6.4.2 Public Order Books Request (PblcOrdrBooksReq)

PblcOrdrBooksReq	
Type:	Inquiry Request
Roles:	<ALL>
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

The Public Order Books Request is used to retrieve the local view of the public order books.

It is possible to request the order book for a dedicated contract in a given delivery area or for all active contracts of a list of products. Either the contract and delivery area or the list of products must be defined in the request.

A Public Order Book Request always returns a Public Order Books Response containing the complete information of the requested order books.

Note that the purpose of the request is to get an initial state of the public order books. Subsequent updates of the public order books are sent using the Public Order Books *Delta* Report message, listing all modified orders. Client applications should process these delta reports to keep their view on the public order books up to date. Any Public Order Books *Delta* Report messages in which the order book revision numbers are smaller than those in the Public Order Book Request must be ignored.

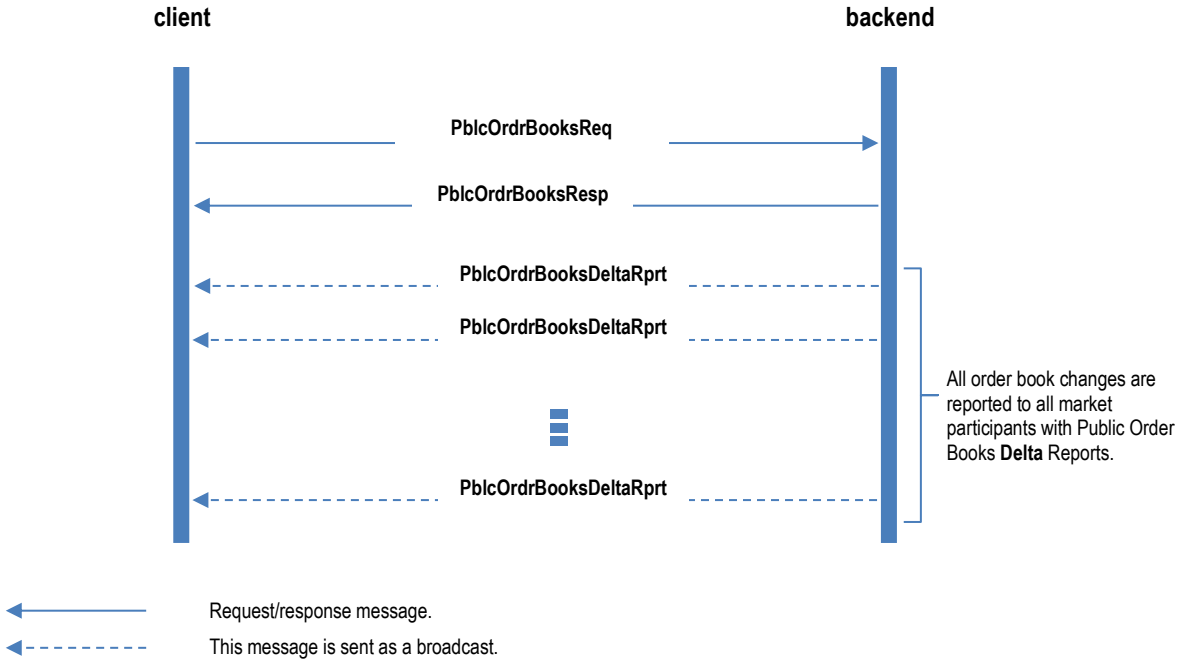


Figure 6: Message flow after a public order book request.

Table 21: Message layout of a Public Order Books Request.

6.4.3 Public Order Books Response (PblcOrdrBooksResp)

PblcOrdrBooksResp	
Type:	Inquiry Response
Response to:	PblcOrdrBooksReq (sent to private response queue see 3.1)
Broadcast:	No
Broadcast Routing Keys:	---
Roles:	<ALL>

The public order book is returned to the client as a result of a Public Order Book Request and gives a complete overview of the requested public order books at the time of the request.

Subsequent changes to the public order books as a result of an order entry, modification, deletion or execution or a change in cross border capacity are reported using a Public Order Books Delta Report. The Public Order Books Response and the Public Order Books Delta Report share the same Message Payload format.

The Public Order Books Response is sent to the private response queue of the requester, see 3.1.

Table 22: Message layout of a Public Order Books Response and a Public Order Books Delta Report.

6.4.4 Public Order Books Delta Report (PblcOrdrBooksDeltaRprt)

PblcOrdrBooksDeltaRprt	
Type:	Broadcast
Response to:	n/a
Broadcast:	Yes
Broadcast Routing Keys:	<schema-version>.prddlvr.<prodName>.<dlvryAreaId>
Broadcast audience	All users with assignment of particular product and delivery area.
Roles:	<ALL>

The Public Order Book Delta Report is sent to the client as a result of any change in the order book as a result of an order entry, modification, deletion or execution or a change in cross border capacity. It contains a list of orders added to the market or changed as a result of the above mentioned actions. A quantity of 0 indicates that the order has to be removed from the order book.

The message layout is shared with the Public Order Books Response (see 6.4.3 above).

6.4.5 Cash Limit Request (CashLmtReq)

CashLmtReq	
Type:	Inquiry Request
Roles:	Trader, Market Operation, Broker, Market Maker, Clearing user
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

The Cash Limit Request is used to retrieve the current cash trading limit. The cash limit is used to limit the open financial risk position of a member. It is calculated on a member level and valid for all traders belonging to that member.

The principal use of the Cash Limit Request is to obtain the cash limit at the start of a session or after a communication breakdown. Subsequent changes to the cash limit are broadcast automatically by the back end. See the diagram below for a typical message flow during a user session for the Cash Limit Request and Cash Limit Response messages.

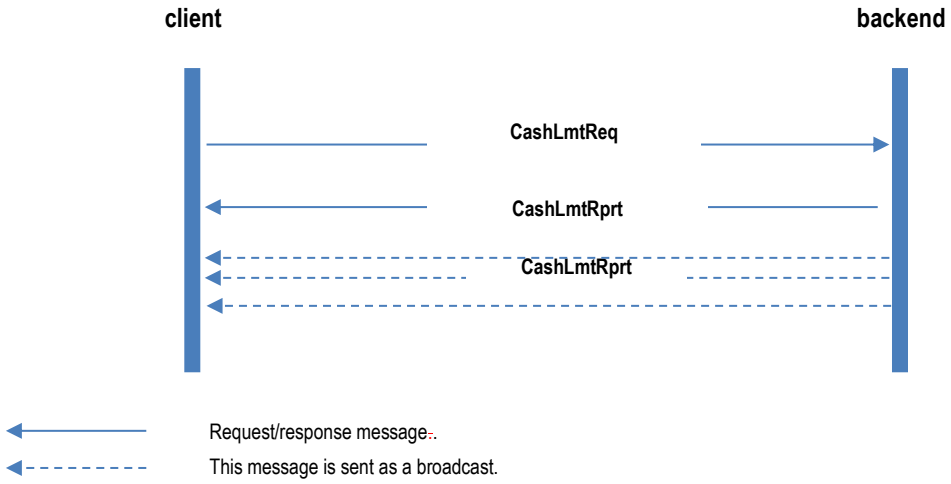


Figure 7: Message flow as a result of a trading limit request.

The trading limit request requires only the member Id as a parameter.

Table 23: Message layout of a Cash Limit Request.

6.4.6 Cash Limit Report (CashLmtRprt)

CashLmtRprt	
Type:	Inquiry Response; Broadcast
Response to:	CashLmtReq (sent to private response queue see 3.1)
Broadcast:	Yes
Broadcast Routing Keys:	`<schema-version>.mbr.<mbrId>`

Broadcast audience:	`All users from particular member`
	`Admins`
	`Brokers with assigned members`
	`Clearing users`

Roles:	`Trader, Market Operation, Broker, Market Maker, Clearing user`

The Cash Limit Report is returned in response to a Cash Limit Request or broadcast as a result of an event that changes the cash limit (Limit creation, modification, deletion or reset) and after house keeping of cash limits (deleting limits that were set up for past dates).

It is not sent during the order management requests (order entry, order execution). See the message flow diagram above (Cash Limit Request).

The message lists all cash limits of the desired Member as well as the current cash limit with its revision.

When the Cash Limit Report is sent as a response it is sent to the private response queue of the requesting user, see 3.1. All subsequent updates are sent as broadcast, so that all traders of the member are up to date.

Table 24: Message layout of a Cash Limit Report.

6.4.7 Cash Limit Delta Report (CashLmtDeltaRprt)

CashLmtDeltaRprt	
Type:	Broadcast
Response to:	--
Broadcast:	Yes
Broadcast Routing Keys:	`<schema-version>.mbr.<mbrId>`

Broadcast audience:	`All users from particular member`
	`Admins`
	`Brokers with assigned members`
	`Clearing users`

Roles:	`Trader, Market Operation, Broker, Market Maker, Clearing user`

The Cash Limit Delta Report broadcasts the new current cash limit value in case the limit changed (e.g. order entry, order execution). All traders/brokers or the Member receive this message.

Table 25: Message layout of a Cash Limit Delta Report.

6.4.8 Commodity Limit Request (CommodityLmtReq)

CommodityLmtReq	
Type:	Inquiry Request
Roles:	Trader, Market Operation, Broker, Market Maker
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

The Commodity Limit Request is used to retrieve the current commodity trading limits of a member. The commodity limit is used to prohibit short selling of a member. The commodity limit is product specific, not all products offer this feature. It is calculated on a member level for every product and valid for all traders belonging to that member.

The principal use of the Commodity Limit Request is to obtain the commodity limits at the start of a session or after a communication breakdown. Subsequent changes to the commodity limit are broadcast automatically by the back end.

The following diagram shows a typical message flow during a user session for the Commodity Limit Request and Commodity Limit Report messages.

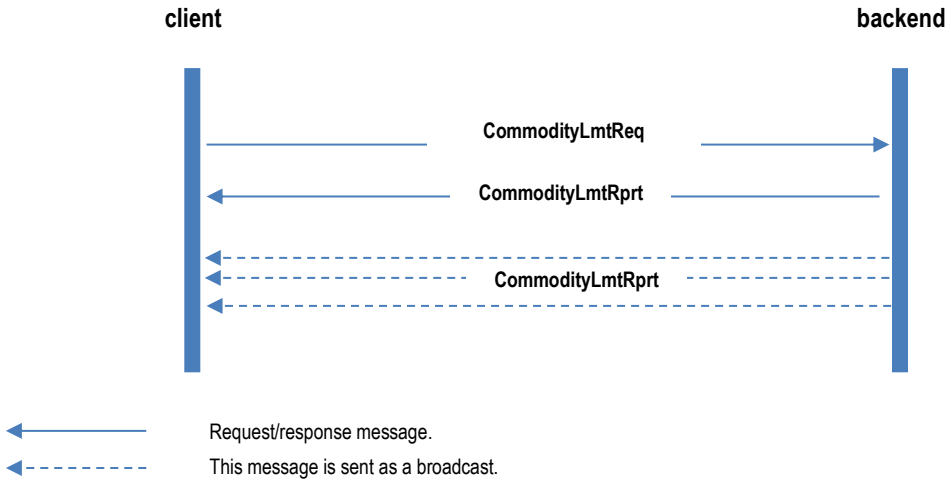


Figure 8: Message flow as a result of a commodity limit request.

The commodity trading limit request requires only the member Id as a parameter.

Table 26: Message layout of a Commodity Limit Request.

6.4.9 Commodity Limit Report (CommodityLmtRprt)

CommodityLmtRprt	
Type:	Inquiry Response; Broadcast
Response to:	CashLimitReq (sent to private response queue see 3.1)
Broadcast:	Yes
Broadcast Routing Keys:	`<schema-version>.mbr.<mbrId>`

Broadcast audience:	`All users from particular member`
	`Admins`
	`Brokers with assigned members`

Roles:	`Trader, Market Operation, Broker, Market Maker`

The Commodity Limit Report is returned in response to a Commodity Limit Request or broadcast as a result of an event that changes the commodity limit (for products eligible for the commodity risk control functionality). See the diagram in the previous subsection (6.4.8 Commodity Limit Request (CommodityLmtReq))

When the Commodity Limit Report is sent as a response it is sent to the private response queue of the requesting user, see 3.1. All updates are sent as a broadcast, so that all traders of the member are up to date.

Table 27: Message layout of a Commodity Limit Report.

6.4.10 Message Request (MsgReq)

MsgReq	
Type:	Inquiry Request
Roles:	<ALL>
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

This inquiry message is used to retrieve the public and/or private messages issued by the backend system in the past.

The principal use of this request is to obtain a list of recent messages at login or after a communication breakdown between the client application and the backend. After login the client application will receive all messages (private and public) automatically as they are broadcast by the backend.

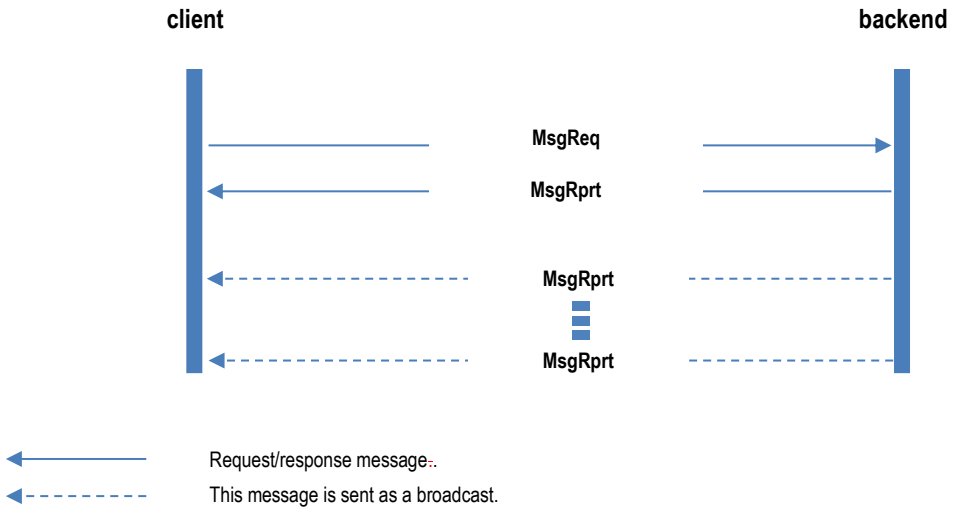


Figure 9: Message flow induced by a message request.

It is required to define a period for which the messages are retrieved and to filter the message type by setting the appropriate value in the type field.

Table 28: Message layout of a Message Request.

6.4.11 Message Report (MsgRprt)

MsgRprt	
Type:	Inquiry Response, Broadcast
Response to:	MsgReq (sent to private response queue see 3.1)
Broadcasted:	Yes
Broadcast Routing Keys:	<schema-version>.prvt.bg.msg.<acctId> <schema-version>.prd.<prodName> <schema-version>.dlvr.<dlvryAreaId> <schema-version>.mkt.<marketAreaId> <schema-version>.mbr.<memberId> <schema-version>.public
Broadcast audience:	Depending on particular message type.
Roles:	<All>

A Message Report is sent as a response to a Message Request to the private response queue of the requesting user, see 3.1.

Additionally, it is broadcasted in the following cases:

- After an order execution.
- In cases of state changes of contracts, delivery areas, market areas and so on.
- User Actions like trade recall request and trade cancellation.
- Admin user sends message to all users.

Table 29: Message layout of a Message Report.

6.4.12 Trade Capture Request (TradeCaptureReq)

TradeCaptureReq	
Type:	Inquiry Request
Roles:	Trader, Market Operation, Broker, Market Maker, Sales
Routing Keys:	m7.request.inquiry

Request Limits:	7/35

This message is used to retrieve trades created during the last five days:

- For *Traders* the account must be specified and all private trades are returned.
- For *Market Operation* user or *Sales* user the account is not needed (a given value will be ignored by the backend) and all trades in the system for the observation period are returned.

The observation period extends from 5 days in the past until the day after the current trading day

Table 30: Message layout of a Trade Capture Request.

6.4.13 Trade Capture Report (TradeCaptureRprt)

TradeCaptureRprt	
Type:	Inquiry Response, Broadcast
Response to:	TradeCaptureReq (sent to private response queue see 3.1)
Broadcasted:	Yes
Broadcast Routing Keys:	<schema-version>.hlftrd.<acctId> <schema-version>.trade.<mktAreaId>
Broadcast audience	Half trade: - Trader (owner of the order) and other traders from his Balancing groups. - Broker with assignment to trader (owner of the order) - Market maker (owner of the order) and other traders from his Balancing groups. Full trade: - Admins - Sales - Settlement users - Clearing users
Roles:	Trader, Market Operation, Broker, Market Maker, Sales

The Trade Capture Report broadcasts private trade information as a result of the following actions:

- Order Execution
- Trade cancelation
- Trade recall request
- Trade recall request processing

The broadcasts are sent to the traders assigned to the accounts for which the related orders were entered and to the market operations users.

Trading participants will always receive half-trades, containing only the information of the order entered by that trading participant:

- If the trading participant entered a SELL order she will see only the SELL side of the trade.
- If the trading participant entered a BUY order she will see only the BUY side of the trade.

The Trade Capture Report is sent to the trading participants with the routing key <schema-version>.hlftrd.<acctId>. Market Operation users will receive the complete trade information (both sell side and buy side). The Trade Capture Report is sent to Market Operation users with the routing key <schema-version>.trade.<mktAreaId>. In the case of a local trade (between 2 delivery areas in the same market area) the message will be sent only once.

When the Trade Capture Report is sent as a reply to the Trade Capture Request, it will be sent to the private response queue of the requesting user (a trade or market operations user), see 3.1.

The Trade Capture Report contains at maximum the trades created during the last five days even if trades for a longer or different period are requested in the Trade Capture Request.

Visibility rules of the data are the same for both the broadcast and response messages.

Table 31: Message layout of a Trade Capture Report.

6.4.14 Public Trade Confirmation Request (PblcTradeConfReq)

PblcTradeConfReq	
Type:	Inquiry Request
Roles:	Trader, Broker, Data Vendor
Routing Keys:	`m7.request.inquiry`
Request Limits:	7/35

This message is used to retrieve a list of public trades executed during a specified time period. This time period can only be 5 days in the past.

Table 32: Message layout of a Public Trade Confirmation Request.

6.4.15 Public Trade Confirmation Report (PblcTradeConfRprt)

PblcTradeConfRprt	
Type:	Inquiry Response, Broadcast
Response to:	PblcTradeConfReq (sent to private response queue see 3.1)
Broadcasted:	Yes
Broadcast Routing Keys:	`<schema-version>.public.trade.<prodName>`
Broadcast audience	`All traders with assignment to traded product` `Admins`
Roles:	Trader, Market Operation

This message is broadcast to all trading participants who are assigned to the product at the event of a trade execution or cancellation or as response to a Public Trade Confirmation Request.

As a broadcast, it contains a list of the public trade confirmations for the triggering event (trade execution, recall or cancellation).

As a response it contains the list of all public trade for the requested product(s) and period. It is sent to the private response queue of the requester, see 3.1.

Pre-arranged trades are not part of the Public Trade Confirmation Report

Table 33: Message layout of a Public Trade Confirmation Report.

6.4.16 Contract Information Request (ContractInfoReq)

ContractInfoReq	
Type:	Inquiry Request
Roles:	<ALL>
Routing Keys:	`m7.request.inquiry`
Request Limits:	7/35

This message is used to retrieve contract related information from the backend system.

If ContractInfoReq with startDate and endDate set, then ACTI, HIBE and IACT contracts are returned where delivery interval or tradingDate are in interval defined by startDate and endDate from request.

If requesting user does not have proper right for requested contract, ErrResp is returned.

Table 34: Message layout of a Contract Information Request.

6.4.17 Contract Information Report (ContractInfoRprt)

ContractInfoRprt	
Type:	Inquiry Response, Broadcast
Response to:	ContractInfoReq (sent to private response queue see 3.1)
Broadcasted:	Yes
Routing Keys:	<schema-version>.prd.<prodName>
Broadcast audience:	All users with assignment to particular product.
Roles:	<ALL>

Returns the detailed information linked to the requested contract(s) as a result of a Contract Information Request, see 3.1.

If ContractInfoReq with startDate and endDate set, then ACTI, HIBE and IACT contracts are returned where delivery interval or tradingDate are in interval defined by startDate and endDate from request.

This message is also broadcasted to indicate contract state changes (i.e. without a prior Contract Information Request).

Table 35: Message layout of a Contract Information Report.

6.4.18 Product Information Request (ProdInfoReq)

ProdInfoReq	
Type:	Inquiry Request
Roles:	<ALL>
Routing Keys:	m7.request.inquiry
Request Limits:	1/10

This message is used to request the product details. It is possible to pass a list of product Ids in one message. If user is requesting a product he does not have right to, ErrResp is returned.

Table 36: Message layout of a Product Information Request.

6.4.19 Product Information Report (ProdInfoRprt)

ProdInfoRprt	
Type:	Inquiry Response
Response to:	ProdInfoReq (sent to private response queue see 3.1)
Broadcasted:	Yes
Broadcast Routing Keys:	<schema-version>.prd.<prodName>
Broadcast audience:	All users with assignment to particular product.
Roles:	<ALL>

This report is used to return detailed product information. It is returned as answer to the Product Information Request and also broadcasted in case of the product modification.

If the prodName in Product Information Request is set then:

For admin role all products with requested prodName(s) are contained in the message.

For other roles the user's balancing group(s) product assignment are checked (see message AccntInfoRprt in chapter 6.6.8).

If the prodName in Product Information Request is not set then:

For admin role all products are contained in the message.

For other roles the products which user's balancing group(s) have assignment are returned (see message AccntInfoRprt in chapter 6.6.8).

In the *ProdCfgs* structure, the following key-value pairs are available:

Prod types:

- FUT: futures
- CRO: cross product spreads
- COM: commodity, energy

Key	Value	Prod Type	Description
isin	true	FUT	The product has an ISIN.
isinValue		FUT	The value of the ISIN.
blockOrderProduct	true	COM	User defined block orders are allowed for this product.
blockMaximumHours		COM	Defines how many base contracts may be grouped together for user defined blocks.
dstBlockProduct	true	COM	Defines the treatment of contracts during 25 hours DST switch.
crossBorderProduct	true	all	The product is traded on the Consolidated Order Book.
icebergOrderProduct	true	all	Iceberg orders are supported for this product.
icebergMinPeakSize	true	all	Minimal peak size of iceberg order
icebergPriceDeltaRange	true	all	Maximal value for peak price delta of iceberg order.
stopOrderProduct	true	FUT, CRO	The product supports stop orders.
indicativeOrderProduct	true	FUT, CRO	Indicative orders are supported for this product.
linkedOrderProduct	true	all	Orders can be linked together for this product.
quoteOrderProduct	true	FUT, CRO	Quotes are supported for this product.
quoteMinQuantity		FUT, CRO	Defines the minimum quantity that a quote is allowed to have.
otcAllowed	true	all	OTC trade registration is allowed for this product.
otcOnly	true	all	The product is for OTC trading only.
onExchangePrearrangedTrade	true	FUT, CRO	The product supports on-exchange prearrange trades.
privateAndConfidential	true	FUT, CRO	Private and confidential trades are allowed for the product.
volatilityInterruption	true	FUT, CRO	Volatility interrupt is relevant for the product.
volatilityPrice		FUT, CRO	Defines the percentage of the price above which volatility interrupt is triggered.
commodityLimitEnabled	true	COM	The commodity limit is enabled for the product.
intraProductSpreads	true	FUT, CRO	The product supports intra-product spreads.
intraProductSpreadContractCount		FUT, CRO	Defines how many contracts are used for intra-product spreads.
productIsWithinDelivery	true	COM	Trading of the product is allowed when delivery has started.
leadTime		COM	Defines the lead time in minutes for products within delivery.
autoOrderMatcher	true	all	The product uses automatcher (regular price-time priority matcher)
liftOrderMatcher	true	FUT, CRO	The product is associated with Hit&Lift matcher.
continuousAONProduct	true	COM	AON orders are supported in continuous trading for this product.
productCommodityId		COM	
referencePrice		all	The initial reference price of the product.
clgHouses	true/false	all	the product matcher has clearing house restrictions
crossProductMatchingEnabled	true	all	The product has cross product matching feature enabled
tradeDecomposition	true	all	The product has trade decomposition feature enabled
groupName		all	Product group name

Table 37: Message layout of a Product Information Response.

6.4.20 Market State Request (MktStateReq)

MktStateReq	
Type:	Inquiry Request
Roles:	<ALL>
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

This message is used to request information about the current market state. In the current version, the message has no content.

Table 38: Message layout of a Market State Request.

6.4.21 Market State Report (MktStateRprt)

MktStateRprt	
Type:	Inquiry Response, Broadcast
Response to:	MktStateReq (sent to private response queue see 3.1)
Broadcasted:	Yes
Broadcast Routing Keys:	<schema-version>.public
Broadcast audience:	All users
Roles:	<ALL>

This message is broadcast when a change in the current market state occurs.

It is also sent as a response to the Market State Request message to the private response queue of the requesting user, see 3.1.

Table 39: Message layout of a Market State Report.

6.4.22 Hub-to-Hub ATC Matrix Request (HubToHubReq)

HubToHubReq	
Type:	Inquiry Request
Roles:	<ALL>
Routing Keys:	m7.request.inquiry
Request Limits:	1/10

This request is used to retrieve the Hub-to-Hub ATC matrix for a given period. To retrieve ATC values, the user must have the "Capacity information" right.

Table 40: Message layout of a Hub-to-Hub ATC Matrix Request.

6.4.23 Hub-to-Hub ATC Matrix Report (HubToHubRprt)

HubToHubRprt	
Type:	Inquiry Response, Broadcast
Response to:	HubToHubAtcReq (sent to private response queue see 3.1)
Broadcasted:	No
Broadcast Routing Keys:	--
Roles:	<ALL>

This message is returned as a response to Hub-to-Hub ATC Matrix Request. It is sent to the private response queue of the user requesting the Hub-to-Hub ATC values, see 3.1.

To retrieve or receive ATC values, the user must have the "Capacity information" right.

Table 41: Message layout of a Hub-to-Hub ATC Matrix Report.

6.4.24 Hub-to-Hub Notification (HubToHubNtf)

HubToHubNtf	
Type:	Broadcast
Broadcasted:	Yes
Broadcast Routing Keys:	<schema-version>.hubToHub
Broadcast audience:	All users with rights to get capacity messages
Roles:	<ALL>

This message is broadcasted in the event of a change (This includes every cross border trade as well as any explicit capacity allocation) to the Hub-to-Hub ATC matrix data and if the minimum waiting time after the last notification has passed. The minimum waiting time is a system parameter that can be configured by the system admin. The parameter can be set in the range of 1to 3 seconds.

Table 42: Message layout of a Hub-to-Hub Notification

6.4.25 Settlement Process Information Request (StlmntProcessInfoReq)

StlmntProcessInfoReq	
Type:	Inquiry Request
Roles:	Market Operation, Sales, Access for Traders is configurable
Routing Keys:	`m7.request.inquiry`
Request Limits:	7/35

A user can send this request to obtain a history of Settlement Process Info Reports. The backend system will respond with a StlmntProcessInfoRprt reply message to the user private response queue, see 3.1.

Note that this is only needed when a user logs in to the system. After login all subsequent settlement process information updates for trades will be automatically broadcast to the user by the backend.

Table 43: Message layout of a Settlement Process Information Request.

6.4.26 Settlement Process Information Report (StlmntProcessInfoRprt)

StlmntProcessInfoRprt	
Type:	Inquiry Response, Broadcast
Response to:	StlmntProcessInfoReq (sent to private response queue see 3.1)
Broadcasted:	Yes
Broadcast Routing Keys:	`<schema-version>.settlement` `<schema-version>.bg.<acctId>`

Broadcast audience:	`Admins, Sales`
	`configurable (with routing key <schema-version>.bg.<acctId>):`
	`- Owner of half of the trade and all users from his balancing group`
	`- Broker with assignment to trader (owner of one of the orders).`

Roles:	Market Operation, Sales, Access for Traders is configurable

This message reports on the various states of settlement processing as they are communicated by the settlement system. It is broadcasted by the backend system to the clients whenever the backend receives updated settlement information.

In addition, this message is sent as a reply message to the private response queue of a trader that sends a Settlement Process Information Request (StlmntProcessInfoReq), see 3.1.

Table 44: Message layout of a Settlement Process Information Report.

6.4.27 Reference Price Update (RefPxUpd)

RefPxUpd	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This message contains the reference price of contracts for a particular date. However the date quoted in the message cannot be greater than current business date.

Every update of a reference price will generate a Reference price report broadcast containing the updated prices.

Table 45: Message layout of Reference price update.

6.4.28 Reference Price Request (RefPxReq)

RefPxReq	
Type:	Inquiry Request
Roles:	<ALL>
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

With this request message it is possible to inquire the reference price for a specific date or all available reference prices of a contract.

- If contractId and date are filled - then the specific reference price is returned in the report, if available.
- If contractId is filled, date is empty - then all available reference prices for the contract are returned in the report.

- contractId is empty, date is filled - then all available reference prices for this date in all contracts assigned to the user are returned in the report.
- If contractId is empty, date is empty- then all available closing prices for all contracts assigned to the user are returned in the report.
- If refPxType is used, only reference prices of this type are returned, if omitted, all reference price types are returned.

Table 46: Message layout of Reference price request.

6.4.29 Reference Price Report (RefPxRprt)

RefPxRprt	
Type:	Inquiry Response; Broadcast
Response to:	Reference price request (sent to private response queue see 3.1)
Broadcast:	Yes
Broadcast Routing Keys:	<schema-version>.prd.<prodName>
Broadcast audience:	All users with assignment to particular product.
Roles:	All

Reference Price report is used as response to Reference Price Request and also to broadcast information after receiving Reference price update message.

Table 47: Message layout of Reference Price Report.

6.4.30 Implied Order Request (ImplOrdrReq)

ImplOrdrReq	
Type:	Inquiry Request
Roles:	Trader, broker, market maker, market operator
Routing Keys:	m7.request.inquiry
Request Limits:	1/10

The Implied order request is used to inquiry for calculated implited orders. If allowed by exchange, the response is ImplOrdrResp. Otherwise ErrResp is returned.

XML Tag	Type	m/o	No.	Data Type	Short description
ImplOrdrReq	SE	m	1	Structure	
StandardHeader	*SE*	*m*		*Structure*	*Standard header of each message. Please see 5.1.3.*
contractID	CE	o	0..n	Long	contract for which the implied orders are requested. If not present, the response will contain implied orders for each contract available to requesting user.

Table 48: Message layout of Implied Order Request Price Report.

6.4.31 Implied Order Report (ImplOrdrRprt)

ImplOrdrRprt	
Type:	Inquiry Response; Broadcast
Response to:	Implied order request (sent to private response queue see 3.1)
Broadcast:	Yes
Broadcast Routing Keys:	<schema-version>.prddlvr.<prodName>.<dlvryAreaId>
Broadcast audicence	All users with assigned product and delivery area
Roles:	Trader, Broker, Market Maker, Market Operator

The Implied order report is used as a response to Implied order request and as a broadcast in case there is a change at the top of orderbook, that causes recalculation of such implied order.

Trader can then match such implied order by using “basket” functionality – sending OrdrEntry message with listExecInst set to IMPL (see 6.2.1) for opposite side.

For details about implied orders, please consult DFS140 – orders.

Table 49: Message layout of Implied Order Report.

6.5 Order Quotes

Order Quotes can be used by market participants to point the market to currently open positions. This feature is configurable in the backend system and might be disabled. The availability can be checked with the capabilities list in the System Info Response (see 6.1.5).

6.5.1 Order Quote Request (OrdrQuoteReq)

OrdrQuoteReq	
Type:	Management Request
Roles:	Trader
Routing Keys:	`m7.request.management`

This message is used to submit an order quote request.

Table 50: Message layout of an Order Quote Request.

6.5.2 Order Quote Report (OrdrQuoteRprt)

OrdrQuoteRprt	
Type:	Broadcast
Response to:	---
Broadcasted:	Yes
Broadcast Routing Keys:	`<schema-version>.<prodName>`
Broadcast audience:	`All users with assigned product.`
Roles:	<ALL>

Broadcast message about an order quote in the market.

Table 51: Message layout of an Order Quote Report.

6.5.3 Order Quote Setup Request (OrdrQuoteSetupReq)

OrdrQuoteReq	
Type:	Management Request
Roles:	Trader
Routing Keys:	`m7.request.management`

This request is used to setup the order quote functionality for one user.

Table 52: Message layout of an Order Quote Setup Request.

6.5.4 Delete Quotes Request (DeleteQuotesReq)

DelQuotesReq	
Type:	Management Request
Roles:	Market Maker
Routing Keys:	`m7.request.management`

Used by market maker for mass deletion of quotes, can be sent on behalf of other user.

The user may send the request for one or several products. For each product in the request, there is the possibility to delete all quotes by setting the Action parameter on the product level to All. If the value of the parameter is None, then only the quotes in the contracts specified in the contract list for this product are deleted. On contract level it is possible to delete either both sides of a quote by setting the actionOnQuoSide parameter to Both on the contract level. It is also possible to delete either the buy side of the quote or the sell side of the quote, by setting the Action parameter to Buy or Sell respectively.

Table 53: Message layout of a Delete all quotes request

6.5.5 Delete Quotes Response (DeleteQuotesResp)

DeleteQuotesResp	
Type:	Management Response; Broadcast
Response to:	DeleteAllQuotesReq (sent to private response queue see 3.1)
Broadcast:	Yes
Broadcast Routing Keys:	`<schema-version>.<prodName>.<dlvryAreaId>.<acctId>`
Broadcast audience:	In case this of on behalf deletion the market maker and the user who has sent the request receive this message as broadcast.
Roles:	`Market maker, (Market Operation trading on behalf)`

The Delete Quotes Response is sent to the user who has entered the Delete Quotes Request. In case this was on behalf deletion the market maker and the user who has sent the request receive the Broadcast message.

If there is an error or more, standard message (MsgRprt) is also sent.

Table 54: Message layout of a Delete Quotes Response.

6.5.6 Quote request (QuoteReq)

QuoteReq	
Type:	Management Request
Roles:	Trader, Market operator trading on behalf
Routing Keys:	`m7.request.management`

The *Quote Request* functionality is used by a trader to request that market makers enter a quote for a specified contract. The request entry is only allowed during *Continuous trading.* Trades can enter Quote Requests for all products that allow quotes.

The request contains the contractId, while the input of a quantity and the selection of the buy/sell side are optional. No input means, that the quote is requested for both sides. Both sides can be sent simultaneously as well.

Public message is broadcasted to market and RfQ report is sent to Market Makers.

Table 55: Message layout of a Request for quote request

6.5.7 Quote response (QuoteResp)

QuoteResp	
Broadcast:	Yes
Broadcast Routing Keys:	`<schema-version>.<prodName>.<dlvryAreaId>.<acctId>`
Broadcast audience	Market Makers that have the relevant product assigned. Admins
Roles:	`Market maker, (Market Operation trading on behalf)`

A Quote Report is triggered as a result to Quote Request. It is broadcasted to Market Makers that have the relevant product assigned.

The report includes all the information from the Quote Request.

Table 56: Message layout of a Request for quote report

6.6 Reference Data

6.6.1 User Report (UserRprt)

UserRprt	
Type:	Management Response, Broadcast
Response to:	LoginReq (sent to private response queue see 3.1
Broadcasted:	Yes
Broadcast Routing Keys:	<schema-version>.mbr.<memberId>
Broadcast audience	Users from member of the affected user
Roles:	<ALL>

The User Report will be returned as a response to a Login Request and broadcasted after any changes done to the user data and user assignments in the backend system.

Table 57: Message layout of a User Report.

6.6.2 Member Change Report (MbrChangeRprt)

MbrChangeRprt	
Type:	Broadcast
Response to:	---
Broadcasted:	Yes
Broadcast Routing Keys:	<schema-version>.public
Broadcast audience	All
Roles:	<ALL>

The Member Change Report is sent for any change done to this Member in the backend system. It is only delivered via Broadcast and cannot be inquired.

Table 58: Message layout of a Member Change Report.

6.6.3 Delivery Area Information Request (DlvryAreaInfoReq)

DlvryAreaInfoReq	
Type:	Inquiry Request
Roles:	<ALL>
Routing Keys:	m7.request.inquiry
Request Limits:	1/10

The Delivery Area Information Request is used to retrieve detailed information about the delivery areas assigned to a particular account. The request may optionally specify one or more products assigned to that account.

Table 59: Message layout of a Delivery Area Information Request.

6.6.4 Delivery Area Information Report (DlvryAreaInfoRprt)

DlvryAreaInfoRprt	
Type:	Inquiry Response, Broadcast
Response to:	DlvryAreaInfoReq (sent to private response queue see 3.1)
Broadcasted:	Yes
Broadcast Routing Keys:	<schema-version>.public <schema-version>.dlvr.<dlvryAreaId>
Broadcast audience	All
Roles:	<ALL>

This message is broadcast whenever a change occurs in an attribute of a delivery area.

In addition it is a response to a Delivery Area Information Request.

Table 60: Message layout of a Delivery Area Information Report.

6.6.5 Market Area Information Request (MktAreaInfoReq)

MktAreaInfoReq	
Type:	Inquiry Request
Roles:	<ALL>
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

The Market Area Information Request is used to retrieve detailed information about the market areas assigned to a particular account. The request may optionally specify one or more products assigned to that account.

Table 61: Message layout of a Market Area Information Request.

6.6.6 Market Area Information Report (MktAreaInfoRprt)

MktAreaInfoRprt	
Type:	Inquiry Response, Broadcast
Response to:	MktAreaInfoReq (sent to private response queue see 3.1)
Broadcasted:	Yes
Broadcast Routing Keys:	`<schema-version>.public` `<schema-version>.mkt.<marketAreaId>`
Broadcast audience	`All`
Roles:	<ALL>

This message is broadcast whenever a change occurs in an attribute of a market area. In addition it is a response to a Market Area Information Request.

Table 62: Message layout of a Market Area Information Report.

6.6.7 Account Information Request (AcctInfoReq)

AcctInfoReq	
Type:	Inquiry Request
Roles:	<ALL>
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

The Account Information Request is used to retrieve the list of accounts in the system. This list can be used as possible counterparties for a pre-arranged order entry.

Table 63: Message layout of an Account Information Request.

6.6.8 Account Information Report (AcctInfoRprt)

AcctInfoRprt	
Type:	Inquiry Response
Response to:	AcctIdInfoReq (sent to private response queue see 3.1)
Broadcasted:	Yes
Broadcast Routing Keys:	`<schema-version>.<acctId>` `<schema-version>.public`
Broadcast audience	`All`
Roles:	<ALL>

This message is returned as a response to an Account Information Request. It is also sent for any change done to an Account in the backend system

Table 64: Message layout of an Account Information Response.

6.6.9 Account Change Report (AcctChangeRprt)

AcctChangeRprt	
Type:	Broadcast
Response to:	
Broadcasted:	Yes
Broadcast Routing Keys:	`<schema-version>.public`
Roles:	<ALL>

The Account Change Report is sent as a result of any change done to an Account in the backend system. It is only delivered via Broadcast and cannot be inquired. It does not contain clearing accounts assignments and assigned Account Ids.

Table 65: Message layout of an Account Change Report.

6.6.10 All Users Request (AllUsersReq)

AllUsersReq	
Type:	Inquiry Request
Roles:	Trader, Market Operation, Broker, Market Maker
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

Request to retrieve all users of the system. Traders are only allowed to retrieve the users of their own member.

Table 66: Message layout of an All Users Request.

6.6.11 All Users Response (AllUsersResp)

AcctIdInfoResp	
Type:	Inquiry Response
Response to:	AllUsersReq (sent to private response queue see 3.1
Broadcasted:	No
Broadcast Routing Keys:	
Roles:	Trader, Market Operation, Broker, Market Maker

This message is returned as a response to an All Users Request.

Note: loginId, orderRegActive, sendMail, mailAddress, sendSMS and mobileNumber attributes are send only to users with Market Operation role.

Table 67: Message layout of an All Users Response.

6.6.12 Clearing Information request (ClgInfoReq)

ClgInfoReq	
Type:	Inquiry Request
Roles:	<ALL>
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

This message is used to request the Clearing Information details. Response to this request is Clearing information report.

Table 68: Message layout of Clearing information request.

6.6.13 Clearing Information report (ClgInfoRprt)

ClgInfoRprt	
Type:	Inquiry Response, Broadcast
Response to:	ClgInfoReq (sent to private response queue see 3.1
Broadcasted:	Yes
Broadcast Routing Keys:	<schema-version>.public
Broadcast audience	All
Roles:	<ALL>

This response is used to return detailed Clearing information. It is returned as response to the Clearing Info Request and also broadcasted, in case there is a change in the clearing information.

Table 69: Message layout of Clearing information response.

6.6.14 Bespoke contract creation request (BespokeContractReq)

BespokeContractReq	
Type:	Inquiry Request
Roles:	<ALL>
Routing Keys:	m7.request.management
Request Limits:	1/10

This message is used to create a new bespoke contract on backend side. As a response a ContractInfoRprt is returned by the M7 backend if the creation of the bespoke contract was successful, otherwise an error (message ErrResp) is returned.

Table 70: Message layout of Bespoke contract creation request.

6.6.15 Risk set information request (RiskSetInfoReq)

RiskSetInfoReq	
Type:	Inquiry Request
Roles:	<ALL>
Routing Keys:	m7.request.inquiry
Request Limits:	1/10

This message serves to retrieve all available or specific set(s) of risk parameters.

Table 71: Message layout of a Risk set information request.

6.6.16 Risk Set information report (RiskSetInfoRprt)

RiskSetInfoRprt	
Type:	Inquiry Response, Broadcast
Response to:	RiskSetInfoReq (sent to private response queue see 3.1
Broadcasted:	Yes
Broadcast Routing Keys:	<schema-version>.public
Broadcast audience	All
Roles:	<ALL>

Risk Set information report is returned in response to a RiskSetInfoReq to a private response queue and broadcasted publicly whenever a set of risk parameters is added, modified and/or deleted. The broadcast contains only risks sets for which something was modified.

Table 72: Message layout of Risk set information response.

7 Admin Requests and Responses

The requests and responses described in this chapter are used for users with administrative privileges such as Market Operation to communicate with the backend.

7.1 Market Information

7.1.1 ATC Data Request (AtcDataReq)

AtcDataReq	
Type:	Inquiry Request
Roles:	Exchange User
Routing Keys:	m7.request.inquiry
Request Limits:	1/10

This message is a request for the ATC data for specific date and the interconnector for which current ATC is being retrieved. ATC Data only for one interconnector can by requested.

Note: The main difference with the public message HubToHubAtcReq is that the HubToHubAtcResp gives ATC values for pairs of delivery areas (outoging delivery area, incoming delivery area), including all routes between them, whereas ATCResp here shows available ATC on individual interconnector (specified by target area and source area).

To retrieve or receive ATC values, the user must have the “Capacity information” right.

The system replies with an ATC Data Response.

The diagram below shows the message flow around ATC data for a typical client session

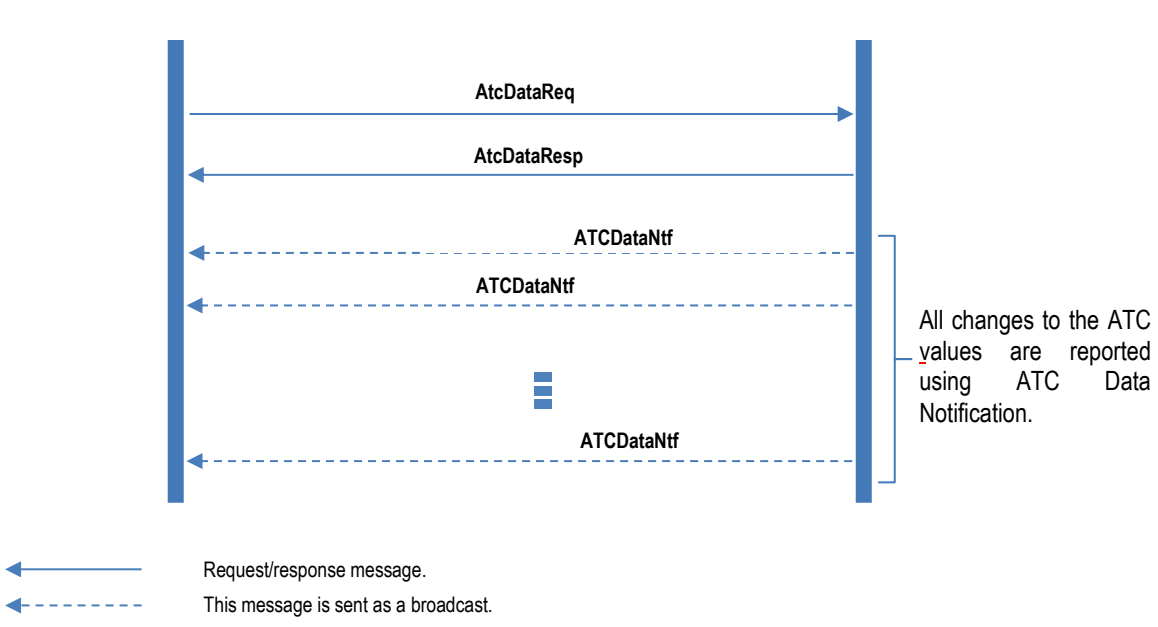


Figure 10: ATC data related message flow for a typical client session

The table below shows the message structure for the ATCDataReq message.

Table 73: Message layout of an ATC Data Request message

7.1.2 ATC Data Response (AtcDataResp)

AtcDataResp	
Type:	Inquiry Response
Response to:	ATCDataReq (sent to private response queue).
Broadcasted:	No
Broadcast Routing Keys:	---
Roles:	Exchange User

This message is returned as a response to an ATC Data Request.

Table 74: Message layout of an ATC Response or ATC Delta Report message

7.1.3 ATC Data Notification (AtcDataNtf)

AtcDataNtf	
Type:	Broadcast
Response to:	
Broadcasted:	Yes
Broadcast Routing Keys:	`<schema-version>.atc`
Roles:	Exchange User

This message is broadcast whenever there is a change in the ATC data. It contains only the new and updated ATC values. ATC values that did not change are not reported.

The message structure is identical to the ATCDataResp message structure in the chapter above.

Every time an update (trade or explicit allocation) happens the difference between the values in the last broadcast and the current values is broadcasted for both direction of the interconnector/border. This can be either one or several hours depending on the type of the allocation and the ramping constraints, if there are any configured.

Note: Please notice that the broadcast information is sent on interconnector level, i.e. the in- and outArea fields are the EIC codes of the Delivery Area. For borders with common ATC, i.e. where the available capacity is common for all interconnectors of the border, this broadcast is sent for the leading interconnector.

7.2 Trade Acknowledgement

7.2.1 Trade Settlement Request (TradeStlmntReq)

TradeStlmntReq	
Type:	Inquiry Request
Roles:	Settlement Operation
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

This request may be sent by a settlement operations user to request a historical list of all trade settlement reports sent to the settlement system that correspond to the specified criteria.

As a response a trade settlement report (TradeStlmntRprt) is returned to the private response queue of the requesting user.

Table 75: Message layout of a Trade Settlement Request.

7.2.2 Trade Settlement Report (TradeStlmntRprt)

TradeStlmntRprt	
Type:	Inquiry Response, Broadcast
Response to:	TradeStlmntReq (sent to private response queue)
Broadcasted:	Yes
Broadcast Routing Keys:	<schema-version>.settlement
Broadcast audience	Settlement users Clearing users
Roles:	Settlement Operation

This message is broadcast to the settlement system as a notification of the executed trades and corresponding flows.

In addition, the TradeStlmntRprt message is sent as a reply to a TradeStlmntReq from a settlement operation user. In this case the message will be sent via private response queue of the requesting user and there will be no need for an acknowledgement of the reception of the TradeStlmntRprt message.

Flow element is only filled in case M7 backend has the flow information for such trade. (For cross border trades and ONLY when connected to capacity service). In case M7 6.0 is connected to XBID, the flow information is only available in XBID.

Table 76: Message layout of a Trade Settlement Report.

7.2.3 Update Settlement Information (UpdateStlmntInfo)

UpdateStlmntInfo	
Type:	Management Request
Roles:	Settlement Operation
Routing Keys:	m7.request.management

This message is sent by the settlement system as a notification of the reception of a trade and flow notification (TradeStlmntRprt). In this case, the stlmntState attribute of the trade must be “ACKN”. The settlement system must send the message to the routing key <m7.request.management>.

After the initial acknowledgement of the reception of the trade and flow information, the settlement system may use this message to send additional information on the settlement progress with the stlmntState set to “INFO”.

The backend will reply with an Error Response (ErrResp) when it receives an “INFO” state before an “ACKN” was received for the same trade revision.

If the backend receives more than one Update Settlement Info message with the “ACKN” state, it will ignore all but the first of those messages.

The backend system expects that the settlement state revision number (stlmntRevisionNo) increases with every UpdateStlmntInfo sent for a particular Trade revision. If the backend receives an UpdateStlmntInfo with a lower or equal stlmntRevisionNo than a previous message for the same Trade revision, it will ignore this update.

Table 77: Message layout of an Update Settlement Information message.

7.3 Reference Data

7.3.1 All Members Request (AllMbrsReq)

AllMbrsReq	
Type:	Inquiry Request
Roles:	Market Operation,Broker
Routing Keys:	m7.request.inquiry
Request Limits:	1/10

Request to retrieve all members of the system.

Table 78: Message layout of an All Members Request.

7.3.2 All Members Response (AllMbrsResp)

AllMbrsResp	
Type:	Inquiry Response
Response to:	AllMbrsReq (sent to private response queue
Broadcasted:	No
Broadcast Routing Keys:	---
Roles:	Market Operation, Broker

This message is returned as a response to an All Members Request.

Table 79: Message layout of an All Members Response.

7.3.3 Create Members Request (CreateMbrsReq)

CreateMbrsReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to create new members in the system. Multiple members can be created with one request.

The system will confirm the reception of the request with an Acknowledgement Response (AckResp) or reject the message with ErrResp.

If errors occur during the creation of a new member, an Error Response (ErrResp) is returned. This ErrResp message will be sent to private response queue.

Successful creation of a member will result in a MbrChangeRprt to be broadcast (see– chapter 6.6.2). For suspended members, the MbrChangeRprt is not sent.

Admin member can not be created using this message (only one is allowed on exchange).

Note: MbrChangeRprt does not contain Address and Contact information. Changes in those columns need to be checked by inquiring All member request (AllMbrsReq).

Table 80: Message layout of a Create Members Request.

7.3.4 Modify Members Request (ModifyMbrsReq)

CreateMbrsReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to modify attributes of existing members in the system. Multiple members can be modified with one request.

The system will confirm the reception of the request with an Acknowledgement Response (AckResp) or reject the message with ErrResp.

If errors occur during the modification of a member, an Error Response (ErrResp) is returned. This ErrResp message will be sent to private response queue.

Successful modification of a member will result in a MbrChangeRprt to be broadcast (see –chapter 6.6.2). For suspended members, the MbrChangeRprt is not sent.

Note: MbrChangeRprt does not contain Address and Contact information. Changes in those columns need to be checked by inquiring All member request (AllMbrsReq).

Table 81: Message layout of a Modify Members Request.

7.3.5 EMS Modify Member Request (EMSModifyMbrReq)

SuspendMbrReq	
Type:	Management Request
Roles:	Clearing user
Routing Keys:	`m7.request.management`

EMS (Emergency Member Stop) modify member request is used to suspend or activate up to 10 members from trading. This suspension or activation triggers further suspension or activation of accounts, traders and deactivation (hibernation) of orders. Only members of type regular and broker can be suspended using this message. When EMS modify member request is used for activating suspended member, all previously deactivated orders are not activated.

The response to the EMS Modify Member Request is EMS Modify Member Response. In addition, the MbrChangeRprt is broadcasted upon EMSModifyMbrReq.

After EMSModifyMbrReq is made, EMSModifyMemberResp is sent and broadcast of Member Change Report (MbrChangeRprt) is performed.

Table 82: Message layout of a Modify Members Request.

7.3.6 EMS Modify Member Response (EMSModifyMbrResp)

EMSModifyMbrResp	
Type:	Management Response
Response to:	EMSModifyMbrReq
Broadcasted:	No
Broadcast Routing Keys:	
Roles:	Clearing user

This message is used as response to EMS Suspend Modify Member Request. It contains the list of members with the corresponding states and errors.

Table 83: Message layout of a Modify Members Request.

7.3.7 Create Accounts Request (CreateAcctsReq)

CreateAcctsReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to create new accounts for a member in the system. Multiple accounts for different members can be created with one request.

The system will confirm the reception of the request with an Acknowledgement Response (AckResp) or reject the message.

If errors occur during the creation of a new account, an Error Response (ErrResp) is returned. This ErrResp message will be sent to private response queue.

Successful creation of a member or members will result in a AcctInfoRprt and AcctChangeRprt to be broadcast (see –chapters 6.6.8 and 6.6.9).

Table 84: Message layout of a Create Accounts Request.

7.3.8 Modify Accounts Request (ModifyAcctsReq)

CreateAcctsReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to modify existing accounts for a member in the system. Multiple accounts for different members can be modified with one request.

The system will confirm the reception of the request with an Acknowledgement Response (AckResp).

If errors occur during the modification of an account, an Error Response (ErrResp) is returned. This ErrResp message will be sent to private response queue.

Successful modification of a member or members will result in a AcctInfoRprt and AcctChangeRprt to be broadcasted (see chapters 6.6.8 and 6.6.9).

It is not possible to change the reference to the member.

Table 85: Message layout of a Modify Accounts Request.

7.3.9 Create Users Request (CreateUsersReq)

CreateUsersReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to create new users for an account in the system. Multiple users for different accounts can be created with one request.

The system will confirm the reception of the request with an Acknowledgement Response (AckResp).

If errors occur during the creation of a new user, an Error Response (ErrResp) is returned. This ErrResp message will be sent to private response queue.

Successful creation of a user will result in a UsrRprt to be broadcast (see –chapter 6.6.1).

For user rights and roles assignment options, please see MFG130.

Admin user has always all product assigned.

Note: UsrRprt does not contain loginId, orderRegActive, sendMail, mailAddress, sendSMS and mobileNumber attributes. To verify these information All Users Response (AllUsersResp) must be inquired by All Users Request (AllUsersReq).

Table 86: Message layout of a Create Users Request.

7.3.10 Modify Users Request (ModifyUsersReq)

ModifyUsersReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to modify existing users of an account in the system. Multiple users for different accounts can be modified with one request.

The system will confirm the reception of the request with an Acknowledgement Response (AckResp).

If errors occur during the modification of a new user, an Error Response (ErrResp) is returned. This ErrResp message will be sent to private response queue.

For user rights and roles assignment options, please see MFG 130.

Admin user has always all product assigned.

Successful modification of a user will result in an UsrRprt to be broadcast (see – chapter 6.6.1).

Table 87: Message layout of a Modify Users Request.

7.3.11 Create Market Area Request (CreateMktAreaReq)

CreateMktAreaReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to create new market areas in the system. Multiple market areas can be created with one request.

The system will confirm the reception of the request with an Acknowledgement Response (AckResp).

If errors occur during the creation of a new market area, an Error Response (ErrResp) will be returned for each market area creation that has produced errors. These ErrResp messages will be sent to private response queue. Successful creation of a market area will result in an MktAreaInfoRprt to be broadcast (see –chapter 6.6.6).

Table 88: Message layout of an Create Market Area Request.

7.3.12 Modify Market Area Request (ModifyMktAreaReq)

ModifyMktAreaReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to modify the attributes of market areas in the system. Multiple market areas can be modified with one request.

The system will confirm the reception of the request with an Acknowledgement Response (AckResp).

If errors occur during the modification of a market area, an Error Response (ErrResp) will be returned for each market area modification that has produced errors. These ErrResp messages will be sent to private response queue.

Successful modification of a market area will result in an MktAreaInfoRprt to be broadcast (see –chapter 6.6.6).

Table 89: Message layout of a Modify Market Area Request.

7.3.13 Create Delivery Area Request (CreateDlvryAreaReq)

CreateDlvryAreaReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to create new delivery areas in the system. Multiple delivery areas can be created with one request.

The system will confirm the reception of the request with an Acknowledgement Response (AckResp).

If errors occur during the creation of a new delivery area, an Error Response (ErrResp) will be returned for each delivery area creation that has produced errors. These ErrResp messages will be sent to private response queue.

Successful creation of a delivery area will result in an DlvryAreaInfoRprt to be broadcast (see–chapter 6.6.4).

Table 90: Message layout of a Create Delivery Area Request.

7.3.14 Modify Delivery Area Request (ModifyDlvryAreaReq)

ModifyDlvryAreaReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to modify delivery areas in the system. Multiple delivery areas can be modified with one request.

The system will confirm the reception of the request with an Acknowledgement Response (AckResp).

If errors occur during the modification of a delivery area, an Error Response (ErrResp) will be returned for each delivery area modification that has produced errors. These ErrResp messages will be sent to private response queue.

Successful modification of a delivery area will result in an DlvryAreaInfoRprt to be broadcast (see – chapter 6.6.4).

Table 91: Message layout of a Modify Delivery Area Request.

7.3.15 Modify Cash Limit Request (ModifyCashLmtReq)

ModifyCashLmtReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

The Modify Cash Limit Request allows changing the cash limit of one or more members. The cash limit for up to 50 members can be changed with one request. Each change can be activated immediately or at the next standard trading limit reset.

Further validations:

If provided, the startDate of the cash limit is the current or a future date

If provided, the endDate of the cash limit is the current or a future date

For modType "MOD", the lmtId is NOT null

For modType "MOD", the version is higher or equal to the current version

For delType "DEL", the lmtId is NOT null

For delType "DEL", the version is higher or equal to the current version

If any of the above validations fails, an Error Response with an error message and its code is returned.

Any cash limit changes for a member that are executed immediately will trigger a Cash Limit Report (see –*chapter 6.4.6*) to be sent to all traders for that member.

If the startDate of the cash limit is null and the Immediate Activation flag is true, the startDate is set to the current date.

If the startDate of the cash limit is null and the Immediate Activation flag is false, the startDate is set to the date of the next cash limit reset.

If the endDate of the cash limit contains the current date, the Immediate Activation flag is set to true.

If the endDate of the cash limit is null, the endDate remains empty.

Please note: If a new limit is uploaded for current day that is lower than current consumption, all member's orders are hibernated.

Table 92: Message layout of a Modify Cash Limit Request.

7.3.16 Modify Cash Limit Response (ModifyCashLmtResp)

ModifyCashLmtResp	
Type:	Management Response
Response to:	ModifyCashLmtReq (sent to private response queue)
Broadcasted:	No
Broadcast Routing Keys:	---
Roles:	Market Operation

This message is returned as a response to a Modify Cash Limit Request. It will contain a list of all requested cash limit changes with a return code for each change request indicating success or failure.

Table 93: Message layout of a Modify Cash Limit Response.

7.3.17 Modify Commodity Limit Request (ModifyCommodityLimitReq)

ModifyCommodityLimitReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

The Modify Commodity Limit Request allows changing the commodity limit for a contract of a member. The commodity limit for up to 50 contract/member combinations can be changed with one request.

If the open sell orders for a member exceed the new commodity limit value for a contract, the commodity limit for that contract and member will not be changed and an error code will be returned in the Modify Commodity Limit Response.

Any commodity limit changes for a member will trigger a Commodity Limit Report (–chapter 6.4.9) to be sent to all traders for that member.

Table 94: Message layout of a Modify Commodity Limit Request.

7.3.18 Modify Commodity Limit Response (ModifyCommodityLimitResp)

ModifyCommodityLimitResp	
Type:	Management Response
Response to:	ModifyCommodityLimitReq (sent to private response queue)
Broadcasted:	No
Broadcast Routing Keys:	---
Roles:	Market Operation

This message is returned as a response to a Modify Commodity Limit Request. It will contain a list of all requested commodity limit changes with a return code for each change request indicating success or failure.

Table 95: Message layout of a Modify Commodity Limit Response.

7.3.19 Full Product Information Report (FullProdInfoRprt)

FullProdInfoRprt	
Type:	Broadcast
Response to:	
Broadcasted:	Yes
Broadcast Routing Keys:	`<schema-version>.fllPrd.<prodName>`
Broadcast audience	`Admins`
Roles:	Market Operation

This message is broadcasted in case a product has been changed. It is used to synchronize products between central M7 instance and local M7 instance. It is not sent to end users.

7.3.20 Create Clearing Account Request (CreateClgAcctReq)

CreateClgAcctReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to create new clearing accounts. Multiple clearing accounts can be created with one request.

The system will confirm the reception of the request with an Acknowledgement Response (AckResp) or reject the message with ErrResp.

If errors occur during the creation of a new Clearing account, an Error Response (ErrResp) is returned. This ErrResp message will be sent to private response queue.

Successful creation of a Clearing account will result in a ClgInfoRprt to be broadcasted (see chapter 6.6.13).

Maximum one ClgInfoRprt and maximum one ErrResp is generated as response to Create Clearing Account Request.

Note: Clearing house information can be obtained by Clearing Information request, product information can be obtained by Product Information Request (See chapter 6.4.18).

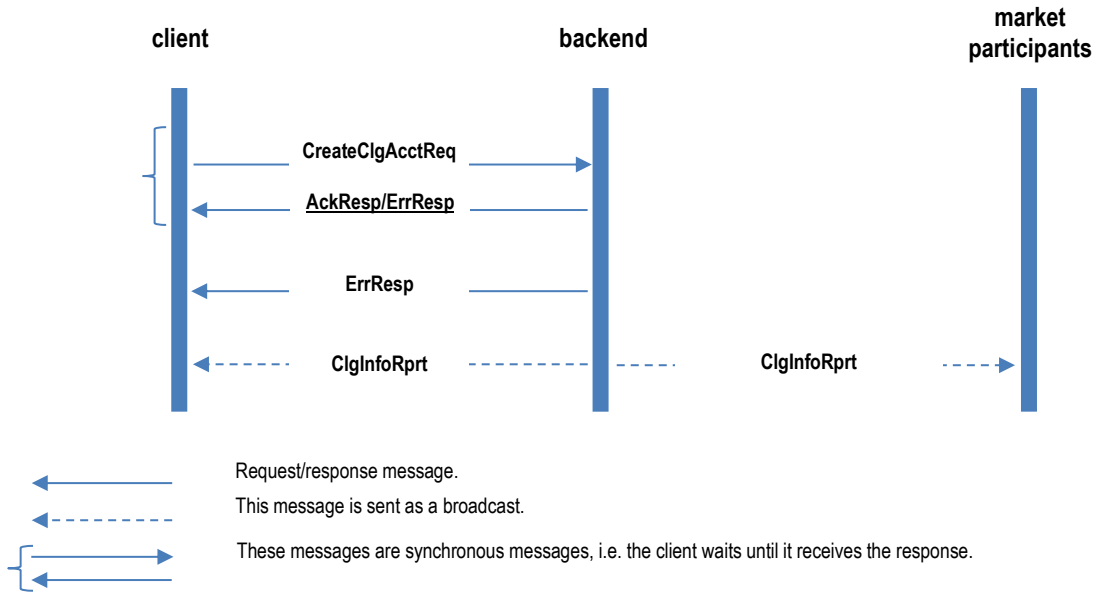


Table 96: Message layout of a Create Clearing Accounts Request.

7.3.21 Modify Clearing Account Request (ModifyClgAcctReq)

ModifyClgAcctReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to modify clearing accounts in the system. Multiple clearing accounts can be modified with one request.

The system will confirm the reception of the request with an Acknowledgement Response (AckResp) or reject the message with ErrResp.

If errors occur during the modification of a Clearing account, an Error Response (ErrResp) is returned. This ErrResp message will be sent to private response queue.

Successful creation of a Clearing account will result in a ClgInfoRprt to be broadcasted (see chapter 6.6.13).

Maximum one ClgInfoRprt and maximum one ErrResp is generated as response to Create Clearing Account Request.

Note: Clearing house and Clearing account information can be obtained by Clearing Information request, product information can be obtained by Product Information Request (See chapter 6.4.18).

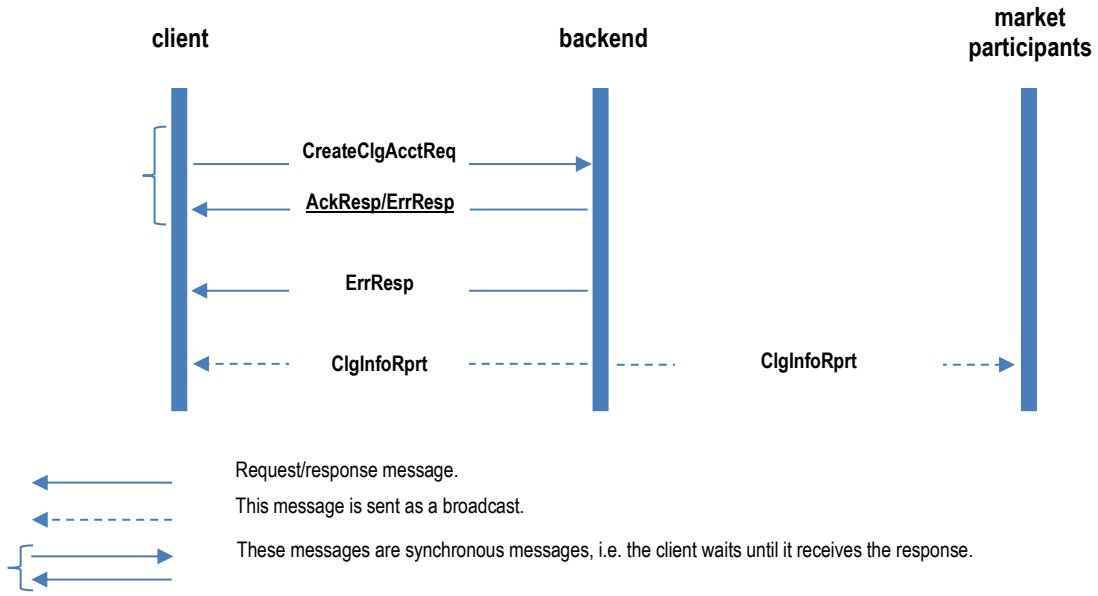


Table 97: Message layout of a Modify Clearing Accounts Request

7.3.22 Create Risk Set Request (CreateRiskSetReq)

CreateRiskSetReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to create new risk set(s). Multiple risk sets can be created with one request.

If creation was succesfull, the RiskSetInfoRprt is broadcasted. If not, ErrResp is returned.

Table 98: Message layout of a Create Risk Set Request.

7.3.23 Modify Risk Set Request (ModifyRiskSetReq)

ModifyRiskSetReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to modify already existing risk set(s). Multiple risk sets can be modified with one request.

If modification was succesfull, the RiskSetInfoRprt is broadcasted. If not, ErrResp is returned.

Table 99: Message layout of a Modify Risk Set Request.

7.3.24 Delete Risk Set Request (DelRiskSetReq)

DelRiskSetReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This request is used to delete already existing risk set(s). Multiple risk sets can be deleted with one request.

If deletion was succesfull, the RiskSetInfoRprt is broadcasted. If not, ErrResp is returned.

Table 100: Message layout of a Delete Risk Set Request.

7.4 Market Operation

The messages described in this section are used to implement market operations specific functionality. They may be used to change the state of a trade or to retrieve full market information.

Only user with a market operations or sales user role will be able to use the requests described in this section.

7.4.1 Trade Recall Processing (TradeRecallProcess)

TradeRecallProcess	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This message is used to handle a trade recall process. This request message is answered by the system with a Trade Processing Response message (see 7.4.3).

Table 101: Message layout of a Trade Recall Processing message.

7.4.2 Trade Cancellation Processing (TradeCancelProcess)

TradeCancelProcess	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

In case of exceptional emergency situations, a market operations user has the possibility to cancel a trade even if no recall request was sent. This request message is answered by the system with a Trade Processing Response message (see 7.4.3).

Table 102: Message layout of a Trade Cancellation Processing message.

7.4.3 Trade Processing Response (TradeProcessResp)

TradeProcessResp	
Type:	Management Response
Response to:	TradeRecallProcess, TradeCancelProcess (sent to private response queue)
Broadcasted:	No
Broadcast Routing Keys:	---
Roles:	Market Operation

The Trade Processing Response message is sent as a response for a Trade Recall Processing request or a Trade Cancellation Processing request and contains the result of the processing.

Table 103: Message layout of a Trade Processing Response.

7.4.4 Full Order Capture Request (FullOrdrCaptureReq)

FullOrdrCaptureReq	
Type:	Inquiry Request
Roles:	Market Operation
Routing Keys:	`m7.request.inquiry`
Request Limits:	1/10

This message is used to retrieve a historical list of all orders entered into the system including the order maintenance steps either triggered by the trader (order modification) or by the backend system (order execution, order inactivation).

As a response to this request a list of Full Order Capture Responses is returned.

Table 104: Message layout of a Full Order Capture Request.

7.4.5 Full Order Capture Response (FullOrdrCaptureResp)

FullOrdrCaptureResp	
Type:	Inquiry Response
Response to:	FullOrdrCaptureReq (sent to private response queue)
Broadcasted:	No
Broadcast Routing Keys:	---
Roles:	Market Operation

The Full Order Capture Response is used to report the maintenance steps of an order. It will list both steps triggered by a user action as well as steps triggered by the backend system.

The message layout of the Full Order Capture Response is shared with the Order Execution Report (see –chapter 6.2.4).

7.4.6 Deactivate Orders For Delivery Area (DeactOrdersForDlvryArea)

DeactOrdersForDlvryArea	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This message is used to request the deactivation of orders for the given delivery areas.

Table 105: Message layout of a Deactivate Orders For Delivery Area message.

7.4.7 Delete Clearing Account Order Request (DelClgAcctOrdrReq)

DelClgAcctOrdrReq	
Type:	Management Request
Roles:	Market Operation
Routing Keys:	`m7.request.management`

This message is used to request a modification of orders with remote Clearing House. It is sent from LOB to COB. As a response to this message is DelClgAcctOrdrResp is sent to LOB.

Table 106: Message layout of Delete Clearing Account Order Request message.

7.4.8 Delete Clearing Account Order Response (DelClgAcctOrdrResp)

DelClgAcctOrdrResp	
Type:	Management Response
Response to:	DelClgAcctOrdrReq
Broadcasted:	No
Broadcast Routing Keys:	---
Roles:	Market Operation

This message is used as response to Delete Clearing Account Order Request. It contains the list of order ids, for which the modification/deletion failed and thus need manual input.

Table 107: Message layout of Delete Clearing Account Order Response message.

7.4.9 Deactivate Orders For Exchange (DeactOrdersForExchange)

DeactOrdersForExchange	
Type:	Management Request
Roles:	Market Operation (exchange user only)
Routing Keys:	`m7.request.management`

The DeactOrdersForExchange is used to deactivate all exchange orders. The order deactivation is handled with high priority by central instance.

The message is processed synchronously, i.e. the ACK response is issued only when the order deactivation is successfully finished.

Only an Exchange user can send the request.

Table 108: Message layout of Deactivate Orders For Exchange message.

8 Message Overview & Access Rights

The following matrix provides an overview of all messages and lists the access rights for different user roles.

Message	Trader	Broker	Market Operation	Market Maker	Sales	Data Vendor	Settlement Operation	Clearing user
General Requests and Responses								
Login Request (LoginReq)	X	X	X	X	X	X	X	X
Logout Request (LogoutReq)	X	X	X	X	X	X	X	X
Logout Report (LogoutRprt)	X	X	X	X	X	X	X	X
System Info Request (SystemInfoReq)	X	X	X	X	X	X	X	X
System Info Response (SystemInfoResp)	X	X	X	X	X	X	X	X
Acknowledgement Response (AckResp)	X	X	X				X	X
Error Response (ErrResp)	X	X	X	X	X	X	X	X
Change password Request (ChgPwdReq)	X	X	X	X	X	X	X	X
Order Entry and Maintenance								
Order Entry (OrdrEntry)	X	X	X	X				
Order Modify (OrdrModify)	X	X	X	X				
Order Request (OrdrReq)	X	X	X	X				
Order Execution Report (OrdrExeRprt)	X	X	X	X				
Pre-Arranged Order Processing (PreArrangedOrdrProcess)	X	X	X	X				
Modify All Orders (ModifyAllOrdrs)	X	X	X	X				
Order Limit Reqest (OrdrLmtReq)	X	X	X	X				
Order Limit Report (OrdrLmtRprt)	X	X	X	X				
Trade Maintenance								
Trade Recall Request (TradeRecallReq)	X	X	X	X				
Prearranged Trade Entry (PrearrangedTradeEntry)		X	X					
Market Information								
Public Order Books Request (PblcOrdrBooksReq)	X	X	X	X	X	X	X	
Public Order Books Response (PblcOrdrBooksResp)	X	X	X	X	X	X	X	
Public Order Books Delta Report (PblcOrdrBooksDeltaRprt)	X	X	X	X	X	X	X	
Cash Limit Request (CashLmtReq)	X	X	X	X				X
Cash Limit Report (CashLmtRprt)	X	X	X	X				X
Cash Limit Delta Report (CashLmtDeltaRprt)	X	X	X	X				X
Commodity Limit Request (CommodityLmtReq)	X	X	X	X				

Message	Trader	Broker	Market Operation	Market Maker	Sales	Data Vendor	Settlement Operation	Clearing user
Commodity Limit Report (CommodityLmtRprt)	X	X	X	X				
Message Request (MsgReq)	X	X	X	X	X	X	X	
Message Report (MsgRprt)	X	X	X	X	X	X	X	
Trade Capture Request (TradeCaptureReq)	X	X	X	X	X		X	
Trade Capture Report (TradeCaptureRprt)	X	X	X	X	X		X	
Public Trade Confirmation Request (PblcTradeConfReq)	X	X				X		
Public Trade Confirmation Report (PblcTradeConfRprt)	X	X				X		
Contract Information Request (ContractInfoReq)	X	X	X	X	X	X	X	
Contract Information Report (ContractInfoRprt)	X	X	X	X	X	X	X	
Product Information Request (ProdInfoReq)	X	X	X	X	X	X	X	
Product Information Report (ProdInfoRprt)	X	X	X	X	X	X	X	
Market State Request (MktStateReq)	X	X	X	X	X	X	X	
Market State Report (MktStateRprt)	X	X	X	X	X	X	X	
Hub-to-Hub ATC Matrix Request (HubToHubAtcReq)	X	X	X	X	X	X	X	
Hub-to-Hub ATC Matrix Report (HubToHubAtcRprt)	X	X	X	X	X	X	X	
Settlement Process Information Request (StlmntProcessInfoReq)	(X)[4]	(X)[2]	X	X	X			
Settlement Process Information Report (StlmntProcessInfoRprt)	(X)[4]	(X)[4]	X	X	X			
Reference Price Update (RefPxUpd)			X					
Reference Price Request (RefPxReq)			X					
Reference Price Report (RefPxRprt)	X	X	X	X				
Implied Order Request (ImplOrdrReq)	X	X	X	X				
Implied Order Report (ImplOrdrRprt	X	X	X	X				
ATC Request (AtcReq)			X					
ATC Response (AtcResp)			X					
ATC Delta Report (AtcDeltaRprt)			X					
Order Quotes								

[4] Access is configurable on backend side fort the role Trader.

Message	Trader	Broker	Market Operation	Market Maker	Sales	Data Vendor	Settlement Operation	Clearing user
Order Quote Request (OrdrQuoteReq)	X	X	X	X				
Order Quote Report (OrdrQuoteRprt)	X	X	X	X				
Order Quote Setup Request (OrdrQuoteSetupReq)	X	X		X				
Mass Quote Request (MassQuoteReq)			X	X				
Mass Quote Report (MassQuoteRprt)			X	X				
Delete Quotes Request (DeleteQuotesReq)				X				
Delete Quotes Report (DeleteQuotesRprt)				X				
Quote request (QuoteReq)	X	X	X	X				
Quote response (QuoteResp)			X	X				
Reference Data								
User Report (UserRprt)	X	X	X	X	X	X	X	
Account Change Report (AcctChangeRprt)	X	X	X	X	X	X	X	
Member Change Report (MbrChangeRprt)	X	X	X	X	X	X	X	
Delivery Area Information Request (DlvryAreaInfoReq)	X	X	X	X	X	X	X	
Delivery Area Information Report (DlvryAreaInfoRprt)	X	X	X	X	X	X	X	
Market Area Information Request (MktAreaInfoReq)	X	X	X	X	X	X	X	
Market Area Information Report (MktAreaInfoRprt)	X	X	X	X	X	X	X	
Account Information Request (AcctInfoReq)	X	X	X	X	X		X	
Account Information Report (AcctInfoRprt)	X	X	X	X	X		X	
All Users Request (AllUsersReq)	X	X	X	X				
All Users Response (AllUsersResp)	X	X	X	X				
Clearing Information request (ClgInfoReq)	X	X	X	X	X	X	X	
Clearing Information report (ClgInfoRprt)	X	X	X	X	X	X	X	
Bespoke contract creation request (BespokeContractReq)		X	X					
Risk Set Information Request (RiskSetInfoReq)	X	X	X	X	X	X	X	X
Risk Set Information Report (RiskSetInfoRprt)	X	X	X	X	X	X	X	X
All Members Request (AllMbrsReq)			X					X

Message	Trader	Broker	Market Operation	Market Maker	Sales	Data Vendor	Settlement Operation	Clearing user
All Members Response (AllMbrsResp)			X					X
Create Members Request (CreateMbrsReq)			X					
Modify Members Request (ModifyMbrsReq)			X					
EMS Modify Member Request (EMSModifyMbrReq)								X
EMS Modify Member Response (EMSModifyMbrResp)								X
Create Accounts Request (CreateAcctsReq)			X					
Modify Accounts Request (ModifyAcctsReq)			X					
Create Users Request (CreateUsrsReq)			X					
Modify Users Request (ModifyUsrsReq)			X					
Create Market Area Request (CreateMktAreaReq)			X					
Modify Market Area Request (ModifyMktAreaReq)			X					
Create Delivery Area Request (CreateDlvryAreaReq)			X					
Modify Delivery Area Request (ModifyDlvryAreaReq)			X					
Modify Cash Limit Request (ModifyCashLmtReq)			X					X
Modify Cash Limit Response (ModifyCashLmtResp)			X					X
Modify Commodity Limit Request (ModifyCommodityLimitReq)			X					
Modify Commodity Limit Response (ModifyCommodityLimitResp)			X					
Full Product Information Report (FullProdInfoRprt)			X					
Create Clearing Account Request (CreateClgAcctReq)			X					
Modify Clearing Account Request (ModifyClgAcctReq)			X					
Create Risk Set Request (CreateRiskSetReq)			X					
ModifyRiskSetRequest (ModifyRiskSetReq)			X					
DeleteRiskSetRequest			X					
Trade Acknowledgement								
Trade Settlement Request (TradeStlmntReq)							X	

Message	Trader	Broker	Market Operation	Market Maker	Sales	Data Vendor	Settlement Operation	Clearing user
Trade Settlement Report (TradeStlmntRprt)							X	
Update Settlement Information (UpdateStlmntInfo)							X	
Market Operation								
Trade Recall Processing (TradeRecallProcess)			X					
Trade Cancellation Processing (TradeCancelProcess)			X					
Trade Processing Response (TradeProcessResp)			X					
Full Order Capture Request (FullOrdrCaptureReq)			X		X		X	
Full Order Capture Response (FullOrdrCaptureResp)			X		X		X	
Deactivate Orders For Delivery Area (DeactOrdersForDlvryArea)			X					
Delete Clearing Account Order Request (DelClgAcctOrdrReq)			X					
Delete Clearing Account Order Response (DelClgAcctOrdrResp)			X					
Deactivate Orders For Exchange (DeactOrdersForExchange)			X					

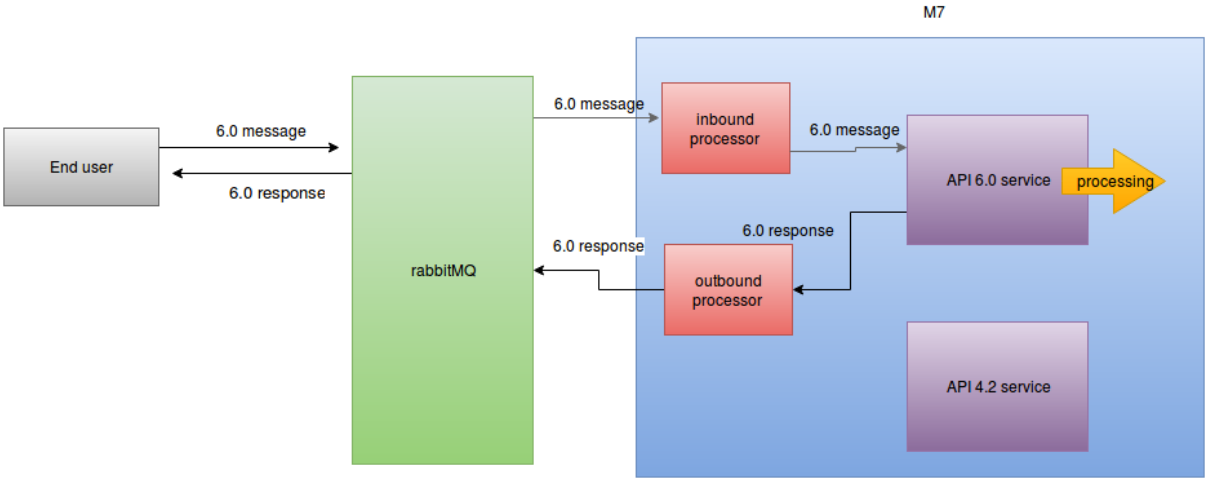
9 Backwads compatibility (4.2 API)

M7 6.0 provides backwards compatibility with 4.2 API for Traders.

9.1 Backwards compatibility principles

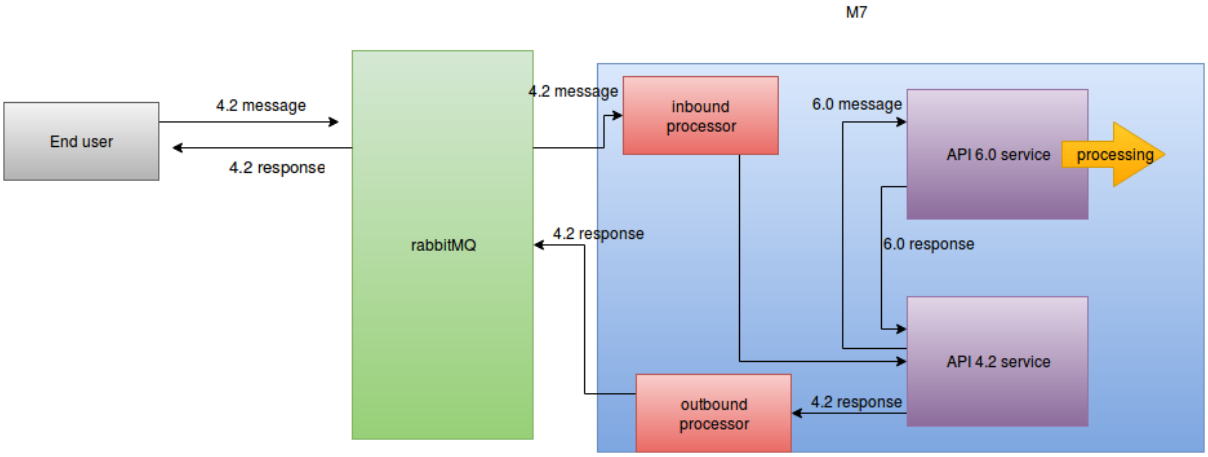
There is equal treatment of messages coming to system using older API version. The principles behind this are shown in the following subchapters.

9.1.1 M7 6.0 API request-response



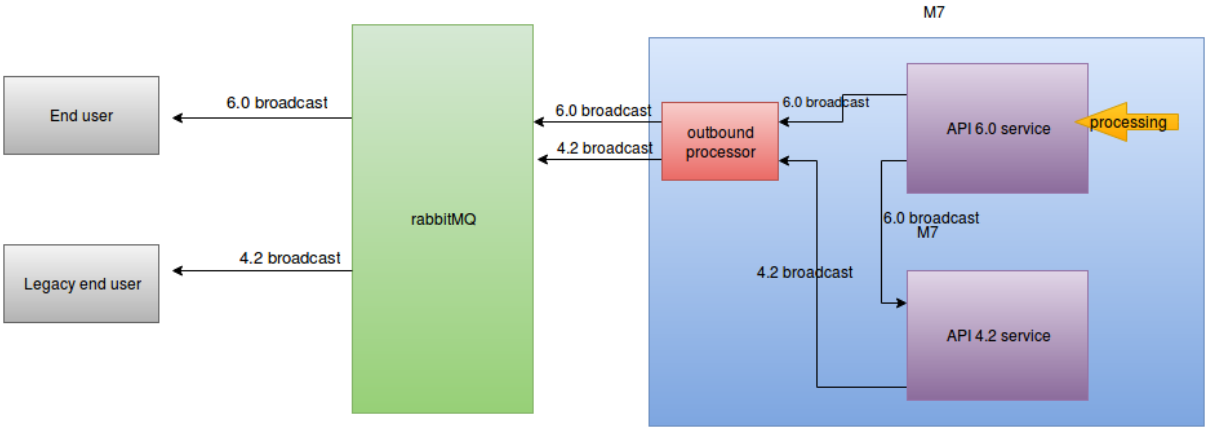
9.1.2 M7 6.0 – API 4.2 request response

API 4.2 requests use the same inbound processor as 6.0 messages. This means, it goes into the same processing queue and the message has the same priority as 6.0 messages (there is no loss of priority for using 4.2 API).



9.1.3 M7 6.0 – API 4.2 Broadcast

4.2 API uses the same outbound processor as 6.0. This means, that 4.2 broadcast immediately follows the 6.0 broadcast and no other message can be inserted between them.



9.1.4 Change requests cross impact

As length of accountIds (balancing group Ids) were changed from char(16) to char(1-32) between version 4.2 and 6.0, this means, that

- Balancing group id longer than 16 characters is trimmed to 16 characters in version 4.2
- Balancing group id shorther than 16 characters is filled to 16 characters using a character "-"

CashLmtRprt only contains limits in EUR. Limits in other currencies are ignored.

CashLmtRprt is only sent when there is existing currentCashLmt in EUR setup on member.

Orders with ordrType Q and W will be skipped over in the OrdrExeRprt, and such reports that end up being empty will be skipped as well

UserRprt.defaultDlvryAreaId is deprecated/removed in 6.0 API In 4.2 API the field contains constant value "DEPRECATED_FIELD"

ContractStateType STBY will be mapped from version 6.0 to version 4.2 as HIBE

ContractPhaseType SDAT will be mapped from version 6.0 to version 4.2 as CONT

TradeStateType RSFA ("Request Sent For Approval to SOB") will be mapped from version 6.0 to version 4.2 as RREQ ("Recall Requested")

10 Forward compatibility

M7 6.0.10 release and API brings new elements for ensuring forward compatibility.

Messages now can contain “<any>” element and “<anyAttribute>” attribute. This allows for new elements and attributes to be added in 6.0 API without changing the respective XSDs and connectors resulting with higher **minor** version messages being compatible with lower **minor** API version.

The “processContents” attribute of <any> and <anyAttribute> element is set to either “lax” or “skip” as there will be no new namespace/additional XSD added to 6.0.10 API. However in further minor releases they may be added.

NOTE: please do not use any additional elements when sending **requests**, unless specificly documented otherwise.

For Java (Jaxb) codes following approach is tested and supported:

Accommodating the use of <any> and <anyAttribute> in JAXB is documented at https://jaxb.java.net

Define the Base class; this class would be implemented by all the classes that need to be Extensible:

```
import java.util.List;
import java.util.Map;
import javax.xml.bind.annotation.XmlAnyAttribute;
import javax.xml.bind.annotation.XmlAnyElement;
import javax.xml.bind.annotation.XmlType;
import javax.xml.namespace.QName;
import org.w3c.dom.Element;


public class BaseSchemaExtensible
{
    @XmlAnyElement(lax=true)
    private List<Element> otherElements; // this will have <any> tag data

    @XmlAnyAttribute
    private Map<QName,Object> otherAttributes; // this will have
<anyattribute> tag data
}



@XmlRootElement
class Person extends BaseSchemaExtensible {

  public String getName();
  public void setName(String);

}
```