

Public Message Interface - FAQs

Creation date : 10/03/2021

Status: Final

Version: 1.1

1. Introduction	4
1.1 Reference	4
1.2 Target Group	4
1.3 Purpose	4
1.4 Public Message Interface Overview	4
2. Questions and Answers	6
2.1 API Processes	6
2.1.1 API Process: from Initial Interest to Go Live	6
2.1.2 API Support Process / Questions	6
2.2 Documentation	6
2.3 Technical Connection, Queues and Messages	6
2.3.1 AMQP and M7 Connectivity	6
2.3.2 TLS Certificates	11
2.3.3 AMQP Queues usage	12
2.3.4 Messages encoding – gzip	12
2.3.5 Request Errors	13
2.4 Understand Requests and broadcasts	13
2.4.1 Sample Requests and broadcasts	13
2.4.2 Inquiry Requests	14
2.4.3 Management Requests	14
2.4.4 Revision Number	15
2.4.5 Broadcasts (including heartbeats)	16
2.5 Recovery procedures	16
2.5.1 AMQP connection or heartbeat loss	16
2.5.2 M7 Heartbeat loss	17
2.5.3 Message Overflow Handling : Unroutable request detected by the Return Listener	17
2.5.4 Gap detection	18
2.5.5 Inconsistent revisionNo	24
2.6 Functional	24
2.6.1 Contracts	24
2.6.2 Order books	27
2.6.3 Orders	31
2.6.4 Trades	31
2.6.5 Balancing Groups	31
2.6.6 Indexes and VWAP	32
2.6.7 Self-trades and Cross trades	32

Historical of the document (Update must be performed after any modification of the process)

Evolution	Date	Version	Author	Comments
Creation	11/08/2017	0.1	Florian Brunzlow	Initial document
Update	31/03/2020	1.0	M7 API Support team	Extension for M7 6.8 based on 2018-2019-Q1 2020 customers questions
Update	22/09/2020	1.04	M7 API Support team	RevisionNo recommended checks 2.4.4
Update	12/10/2020	1.05	M7 API Support team	a) Mistake in public trades routing key format, aligned on DFS180: <schema-version>.pbic.trd.<prodName> b) False Gap detection
Update	10/03/2021	1.1	M7 API Support team	Precision in the contract name logic in API messages

Abbreviations and Terminology

AMQP	Advanced Message Queuing Protocol
Account ID	API term for an M7 Balancing Group. Can be functionally assimilated to a portfolio
Client	Program or application acting as a client of the AMQP server
Documentation	This refers to the following documents: DFS180 (M7 API specifications), DFS200 (M7 API messages), M7 API Presentation slides, and all other documents of the API package
FAQ	Frequently Asked Questions
Gap	Discontinuity in the broadcast messages sequence for a given routing key
Trader	Person that logs into the client to participate in the market
Login-ID	Traders login ID for the M7 System
Routing Key	<p>A way of implementing broadcasting rules. A single message or several messages can correspond to a given routing key.</p> <p>A given broadcast is distributed to API users with assignments matching the broadcast routing key (e.g. assigned to a given product and delivery area for Public order books messages).</p>
MO	Market Operations
OBK	Order Book
Qty	Quantity

1. Introduction

1.1 Reference

This document is an amendment of the DFS180 M7 Public API specification. It takes over topics and terminology of this specification document. In case more information is needed regarding the items hereafter please refer to the above-mentioned document.

To avoid duplicating too much content, many references are also done to the M7 API Presentation slides, which contain many diagrams illustrating the below topics.

All these documents can be found in the API Package that can be provided again if required by Market Operations.

1.2 Target Group

This document is addressed to the members and API customers of SEMOpX.

1.3 Purpose

The purpose of the document is the provision of answers to frequently asked questions regarding the M7 Service Public Message Interface (commonly called “M7 API”). In case the provided answers would not help the customer, he can contact the Market Operations team as the First Level Support.

1.4 Public Message Interface Overview

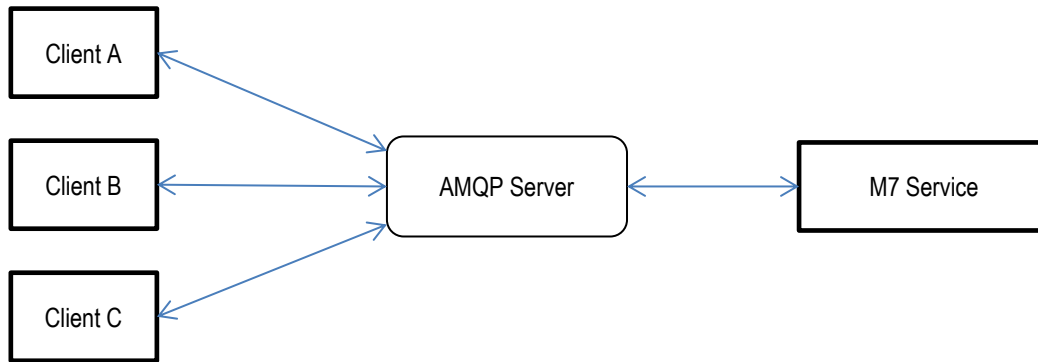
The M7 Public Message Interface allows clients to communicate with the M7 System via a programmable interface.

The communication with the M7 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. The M7 System is currently using 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.

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

Clients using the Public Message Interface communicate with an AMQP server (or “broker”), which in turn communicates with the M7 Service. The M7 Service itself is behind a firewall and not directly accessible by customers.



Overview of communication between clients and the M7 Service

For messaging between the clients and the M7 Service, two basic patterns are supported:

- Request-response communication (initiated by the client),
- and broadcast communication (initiated by the M7 Service).

All communication is encrypted using Transport Layer Security 1.2 (TLS) protocol. 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.

2. Questions and Answers

2.1 API Processes

2.1.1 API Process: from Initial Interest to Go Live

Q: What are the different steps before I can go live with my M7 API application?

A1: Please refer to the diagram “From interest to Go-Live” in the first slides of the M7 API presentation.

A2: You should register for API access with SEMOpx Registration.

A3: Once you received everything you need, you can start developing your software in advanced simulation environment.

A4: Before you can go live with your software, you will need to pass a 24h conformance test.

2.1.2 API Support Process / Questions

Q: Who should I write to when I have an M7 API related question?

A1: EPEX SPOT SEMOpx Operator <marketops@ops.semopx.com>

You are of course very welcome to give us a call to accompany your email, especially in case of emergency. Emails are usually required to get technical details (logs, exact error messages, etc.).

2.2 Documentation

Q: How can I access the M7 API documentation? (latest M7 API package)

A1: Downloadable from the SEMOpx Website.

Q: Where can I get more information about AMQP?

A1: Please see the following links: <http://www.amqp.org>, <https://www.rabbitmq.com/>

Q: What is the difference between AMQP and Rabbit MQ?

A: Rabbit MQ is an open source message broker company that implemented the Advanced Message Queuing Protocol (AMQP), installed on the M7 API servers. AMQP is often the term used to actually simply refer to Rabbit MQ (e.g. “the AMQP server”).

2.3 Technical Connection, Queues and Messages

2.3.1 AMQP and M7 Connectivity

Q: Where can I find M7 environments details?

A: It can be downloaded from the SEMOpx website

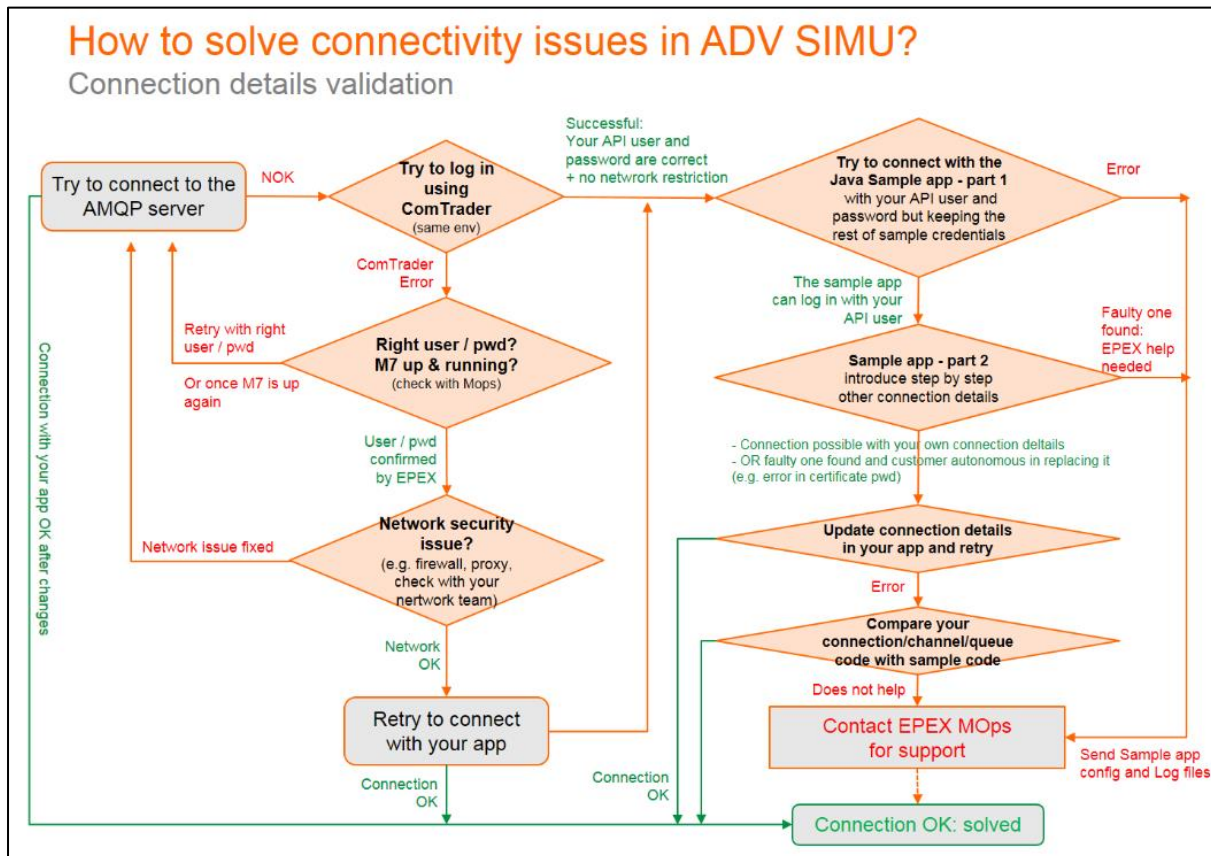
Q: Where can I find my M7 connection details (user and password, application ID, certificate and certificate password)?

A1: If you are a member please just write an email to SEMOpx Market Operations

Q: I am not able to establish a connection with the AMQP Server (API server).

A: For Test environments (ADV SIMU):

Please follow the below process and steps explanations:



Step 1: Verify if you are able to log in with your API user and password using the SEMOpX Trading client ComTrader for the same environment (refer to the Environment Details document on our website).

ComTrader is a Java API application:

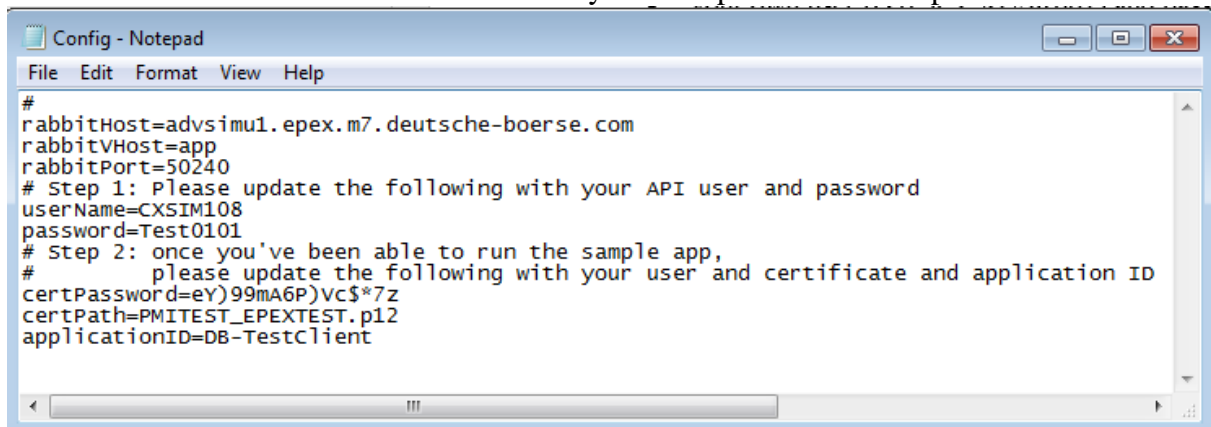
- if a connection using ComTrader is possible, this means that:
 - o your API user and password are valid
 - o an issue in your company or the service provider's infrastructure can be excluded (i.e. proxy settings, firewall settings),
 - o As a result the problem is located:
 - either in the parameters your app is using to try to connect (please double check them in the Environment Details document on our website) :
 - Advance Simulation example:
 - rabbitHost1 = advsimu1.epex.m7.deutsche-boerse.com
 - rabbitHost2 = advsimu2.epex.m7.deutsche-boerse.com
 - rabbitVHost = app
 - rabbitPort = 50240
 - API user Name = SX.....
 - API user password = ...
 - certPassword = ...
 - certificate name and path = PMITEST_EPEXTEST.p12
 - application ID = ...
 - OR in your API application implementation:
 - e.g. your app does not respect the private response queue naming convention indicated in the DFS180. **Please note that M7 6.12 will introduce a mandatory**

new naming convention to limit the number of private response queues to 10 (already described in DFS180 M7 6.11)

- Note: to launch ComTrader, Java version 8 update 60 or higher is supported
- If a connection via ComTrader is not possible, please verify:
 - If you have a user / password related error message please double check with market operations that you are using the right credentials, or use the ComTrader Login window “Forgotten password” to receive a new one by email (please refer to the “Reset password” section below)
 - If you cannot get a new password, please contact Market Operations.
 - if your company’s firewall allows network traffic on the used port.
 - with Market Operations if the environment is up and running.
 - otherwise check the IP-address (URL) and the port used to connect to the M7 AMQP server.

Step 2: Verify if it is possible (only in a TEST environment) to log in using the Java sample executable app provided in the API package:

- Step 1: just replacing in the config file the User name and password with your credentials: if you could connect to CT then the combination of your user/pwd and default parameters should work.



```
#
rabbitHost=advsimu1.epex.m7.deutsche-boerse.com
rabbitVHost=app
rabbitPort=50240
# Step 1: Please update the following with your API user and password
userName=CXSIM108
password=Test0101
# Step 2: once you've been able to run the sample app,
# please update the following with your user and certificate and application ID
certPassword=eY)99mA6P)VC$*7z
certPath=PMITEST_EPEXTEST.p12
applicationID=DB-Testclient
```

- Step 2: update the second part:
 - your certificate and its password (for the right environment),
 - if you can log in this means your certificate/pwd combination is valid
 - if you cannot please report this certificate/pwd issue to Market operations.
- Step 3: update your application ID
 - If you can log in this means all your connection parameters are ok.
 - If you cannot please report this application ID issue to Market operations.

Step 3: if the previous steps enabled you to validate that all your connection parameters are correct (you were able to log in ADV SIMU with the sample app configured with your connection details), and if comparing your code with the connectivity part of the sample code or if DFS180 could not help please report your issue as explained in the above 2.1.2 section.

Q: I'm not able to start ComTrader.

A1: Verify that the Java Runtime Environment (version 8 update 60 or higher) is installed.

Q: What are the pre-requisites to be able to log in M7?

A: Your API app must be already connected to the AMQP server and have created a private response queue, so that it is able to read the response sent by M7 to the Login request (User Report or Error Response).

The private response queue should be created as :

- **Durable** : the queue will survive a broker restart, it is ok to have it there, no impact on queue deletion.
- **Auto-delete** : should delete the queue when explicitly asked (consumer.cancel) or in case of a closed channel or connection, or lost TCP connection with the server.
- **Exclusive**: used by only one connection and the queue will be deleted when that connection closes

Q: Connection to the AMQP server is possible, but I'm not able to log in M7.

A1: Verify your user id is the one sent by Market Operations (e.g. SXUSER01).

A2: Ensure your Login Request looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<LoginReq xmlns="http://www.deutsche-boerse.com/m7/v6" user="CXSIM108" force="true"
disconnectAction="NO">
  <StandardHeader marketId="EPEX"/>
</LoginReq>
```

A3: Please make sure all these mandatory properties in your message headers are as well filled in:

- correlationId
- replyTo
- userId
- appId
- contentType

A4: In particular ensure **contentType = 6.0**

- The value must be 6.0 even if you are using a more advanced 6.x API schema (e.g. 6.8). It corresponds to the last API schema major version.
- If sending anything else than 6.0, your API application will not even receive any User Report as a response to your LoginReq. No broadcast would be received either, and thus no heartbeat.

Q: How to use the *force* attribute in the Login Request?

The *force* attribute is a flag that indicates if the user trying to log in wants to force a login even if an user with the same credentials is already logged in into the backend system.

Basically, if the same user is already logged in:

- when sending a Login Req with *force*= "false", M7 answers an ErrRresp. The already logged in user remains connected.
- when sending a Login Req with *force*= "true", M7 logs out the already connected user and enables the new request to log in (and answers a User Report).
 - **Caution:** in that case, the application that has just been kicked out received a Logout Report: **the logged/kicked out application should absolutely close the AMQP connection** so that it does not continue to consume the broadcast queue ; otherwise there would be 2 consumers stealing messages from each other, leading to an inconsistent set of messages (each app would see missing messages, resulting in Sequence ID gaps).

The main purpose of this feature is to enable applications with a GUI not to systematically logout/kick out another instance of the application that would already be connected with the same user.

Here is an example anyone can reproduce using the ComTrader API application:

1. Initial condition: user CXSIM108 logs in with ComTrader 1 (was not logged in before). Since login is successful it receives a User Report response:

```
<?xml version="1.0" encoding="UTF-8"?>
<UserRprt xmlns="http://www.deutsche-boerse.com/m7/v6">
  <StandardHeader marketId="EPEX"/>
  <Usr sessionId="1551291300179" mbrName="Test Member 001" usrId="1629" revisionNo="4" name="Test User
Jean-Marc" usrCode="TRD001" mbrId="TM001" defaultAcctId="11XTM1-TM-----1" state="ACTI">
    <AssgAcctId>TT01-UK--MIDP</AssgAcctId>
    <AssgAcctId>11XTM1-TM-----2</AssgAcctId>
    <AssgAcctId>11XTM1-TM-----1</AssgAcctId>
    <UsrRole>ROLE_CAPACITY_INFO</UsrRole>
    <UsrRole>ROLE_SYSTEM_ACCESS</UsrRole>
    <UsrRole>ROLE_PUBLIC_API</UsrRole>
    <UsrRole>ROLE_USER</UsrRole>
  </Usr>
</UserRprt>
```

2. Another app (ComTrader 2) sends a Login Request with Force = False for the same user:

```
<?xml version="1.0" encoding="UTF-8"?>
<LoginReq xmlns="http://www.deutsche-boerse.com/m7/v6" user="CXSIM108" force="false"
disconnectAction="NO">
  <StandardHeader marketId="EPEX"/>
</LoginReq>
```

3. Because the user I am trying to connect with is already logged and force = false the ComTrader 2 app receives an ErrRsp message:

```
<?xml version="1.0" encoding="UTF-8"?>
<ErrResp xmlns="http://www.deutsche-boerse.com/m7/v6">
  <StandardHeader marketId="EPEX"/>
  <Error err="&quot;TraderAlreadyLoggedInException [traderId=CXSIM108,
previousLoginTimestamp=1,551,982,565,210, Amqp]&quot;" errCode="8000">
    <VarList>
      <Var id="0" value="CXSIM108"/>
      <Var id="1" value="1551982565210"/>
      <Var id="2" value="Amqp"/>
    </VarList>
  </Error>
</ErrResp>
```

- This is used by applications with a GUI to ask a confirmation to the end user: does he want to kick out the connected user or not?

4. But when ComTrader 2 tries to login directly with force = true then the already with ComTrader 1 connected user is automatically kicked out and receives a logout report:

- ComTrader 2:

```
<?xml version="1.0" encoding="UTF-8"?>
<LoginReq xmlns="http://www.deutsche-boerse.com/m7/v6" user="CXSIM108" force="true"
disconnectAction="NO">
  <StandardHeader marketId="EPEX"/>
</LoginReq>
```

- ComTrader 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<LogoutRprt xmlns="http://www.deutsche-boerse.com/m7/v6" usrId="1629" sessionId="1551291300179"
forced="true">
  <StandardHeader marketId="EPEX"/>
</LogoutRprt>
```

- ComTrader 2 receives its User Report as a login confirmation, with its own session Id:

```
<?xml version="1.0" encoding="UTF-8"?>
<UserRprt xmlns="http://www.deutsche-boerse.com/m7/v6">
  <StandardHeader marketId="EPEX"/>
  <Usr sessionId="1551291300180" mbrName="Test Member 001" usrId="1629" revisionNo="4" name="Test User
Jean-Marc" usrCode="TRD001" mbrId="TM001" defaultAcctId="11XTM1-TM-----1" state="ACTI">
    <AssgAcctId>TT01-UK--MIDP</AssgAcctId>
```

```
<AssgAcctId>11XTM1-TM-----2</AssgAcctId>
<AssgAcctId>11XTM1-TM-----1</AssgAcctId>
<UsrRole>ROLE_PUBLIC_API</UsrRole>
<UsrRole>ROLE_SYSTEM_ACCESS</UsrRole>
<UsrRole>ROLE_USER</UsrRole>
<UsrRole>ROLE_CAPACITY_INFO</UsrRole>
</Usr>
</UserRprt>
```

Q: Why and when does my application receive a Logout Report for “INACTIVITY”

A1: M7 sends a Logout report with text = “INACTIVITY” when it detects that an API application has closed its AMQP connection without having sent an explicit Logout Request.

- This detection is done after a random duration (from 20 to 40 seconds).
- This usually happens when an API application loses the AMQP connection (e.g. network glitch) or if the app reacts to an “instability” event (e.g. gap in broadcast message, M7 heartbeat loss, timeout reached for an inquiry request) and closes the AMQP connection without having sent a Logout Request for the current Session ID, that we will call here SESSION_ID_1.
- Since no logout request was sent by the app, the user broadcast queue is not deleted by M7 and keeps on storing messages while the app is not connected anymore.
- When the app recreates an AMQP connection and logs in again, receiving a SESSION_ID_2 in the User Report message, it sooner or later consumes this Logout Report for SESSION_ID_1 (whether sent during the disconnection time or after the relogin): it is just the confirmation that M7 logged out the previous Login session and this message should be ignored as long as the session ID is not the current one.

In order to limit the number of Logout Reports received for INACTIVITY, API apps should always:

- send a Logout Report for the current Session ID before closing the AMQP connection.
- When reading a Logout Report, ignore it if the session ID is not the last active one (the one of the latest User Report)

A2: M7 sends a Logout Report for INACTIVITY when it detects some kind of “imbalance” at the AMQP objects level, that is when an API application closes a channel while the AMQP connection and other channels still exist.

Q: I cannot establish a connection with the API server, what should I do?

RabbitMQ.Client.Exceptions.AuthenticationFailureException : ACCESS_REFUSED - Login was refused using authentication mechanism PLAIN. For details see the broker logfile.

A: You need to reset your API user password.

You can either:

- do this autonomously using ComTrader:
 - Launch ComTrader and click on the “Forgotten password” link in the Login panel
 - M7 will generate a new password and send it by email to the email address registered at the API user level in M7.
- contact Market operations.

2.3.2 TLS Certificates

Q: How long is a certificate valid in PROD or test environments?

A1: The certificates are valid for 3 years.

EPEX SPOT reminds its customers the upcoming expiry and sends new certificates and passwords a few weeks before the expiry.

Q: How can I test that my certificate for a test environment is still valid?

A: Please refer to the “AMQP and M7 Connectivity” section to check the different means.

Q: Why do I have the following error when trying to connect to the AMQP broker from my API client (in C#/NET):

“The remote certificate is invalid according to the validation procedure.”?

A1: Your app needs to use `factory.Ssl.AcceptablePolicyErrors = True` so that it will accept the error and connects (no matter what error occurs).

A2: If you want to be stricter on accepting some errors or not, then you can use the following:

```
factory.Ssl.Version = Security.Authentication.SslProtocols.Tls12
factory.Ssl.AcceptablePolicyErrors =
    Net.Security.SslPolicyErrors.RemoteCertificateNameMismatch Or
    Net.Security.SslPolicyErrors.RemoteCertificateChainErrors Or
    Net.Security.SslPolicyErrors.RemoteCertificateNotAvailable
```

2.3.3 AMQP Queues usage

Note: for all queues related questions please refer to the M7 API presentation slides and DFS180 for further details.

Q: How to start my API application? In which order should I create queues and send requests?

A1: Please refer to the M7 API presentation detailing the proper starting sequence: “Proper Login/start Procedure”.

A2: Please check as well the Java sample code in the API package, that illustrates this.

Q: How many queues should my API app create?

A1: Only the private response queue needs to be created. Please ensure it is created before sending the Login Request.

But please keep in mind that 2 queues will be used:

1. Private response queue: has to be created before your app sends a LoginRequest (to be able to read the response)
2. Broadcast queue: will be created automatically after successful login

Please refer to “AMQP Queues basics” in the API presentation for more details.

Private response queues M7 6.11/6.12 update: Please note that M7 6.12 will introduce a mandatory new naming convention to limit the number of private response queues to 10 (already described in DFS180 M7 6.11)

Q: How can I be sure that the broadcast queue is already created when subscribing to it?

A1: In case of successful login, the broadcast queue is created automatically (`m7.broadcastQueue.<login-id>`) just before M7 sends out a User Report message.

Q: What is meant by subscribing?

A1: This means registering a consumer on the channel to be able to consume messages from a queue. Please refer to the Rabbit MQ website for more details and to the Java sample code.

2.3.4 Messages encoding – gzip

Q: My app cannot read a response or a broadcast content. What should be done?

A: There is a message property that indicates if the message needs to be decompressed or not:

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): <ul style="list-style-type: none"> ▪ 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)

- Your application should be able to detect dynamically at execution time which broadcasts are zipped or not, based on the encoding type.
- Indeed, the list of compressed messages is likely to change in future releases, for performance optimization purposes.
- With M7 current version the User Response is compressed: you need to decompress it to be able to check that you are well logged in.
- Please refer to the Java sample to see an example of decompression implementation and check in the log file all decompressed messages content, including which ones have a gzip encoding content.

2.3.5 Request Errors

Q1: when sending a request I receive an “unknown request” error

A1: Ensure the request is sent with the proper routing key:

- management request with routing key = m7.request.inquiry
- or an inquiry request with RK = m7.request.management

2.4 Understand Requests and broadcasts

2.4.1 Sample Requests and broadcasts

Q1: How can I get sample requests.

A1: please use ComTrader as follows:

- launch ComTrader
- Display the “Recorded Messages” panel using the shortcut CTRL+ALT+R.
- Log in:
 - you can see all the inquiry requests sent by ComTrader during its starting/initialization phase and all M7 responses.

- Once the previous phase is completed you can see as well broadcast messages sent by the M7 API in real time (contract closure, new order in public order book, public trade, etc.)
- If you have a read/write trader access you can as well send management requests by managing:
 - orders (creation, modification, cancellation, deactivation, activation, multiple order entry via the basket, etc.) and trades (recall request).
 - All related management requests, responses and broadcasts will be visible in the “Recorded Messages” panel
- Please click on the magnifier in the XML column to display the XML payload
- Several message key properties such as messages *Routing Key* and *Correlation Id* can be observed in this panel

Client Date	Process Date	RTT	Direction	Routing Key	Correlation ID	Request Type	XML
23.05.2018 15:37:25	23.05.2018 15:37:25	32	➔	6_0.prddlvr.XBID_Hour_Power.10YDE-EON-----1	48eac199-516a-	PbldrOrdBooksDeltaRprt	Q
23.05.2018 15:37:25	23.05.2018 15:37:25	29	➔	6_0.prddlvr.XBID_Hour_Power.10YDE-VE-----2	48eac199-516a-	PbldrOrdBooksDeltaRprt	Q
23.05.2018 15:37:25	23.05.2018 15:37:25	28	➔	6_0.prddlvr.XBID_Hour_Power.10YDE-RWENET---I	48eac199-516a-	PbldrOrdBooksDeltaRprt	Q
23.05.2018 15:37:25	23.05.2018 15:37:25	26	➔	6_0.prddlvr.XBID_Hour_Power.10YAT-APG-----L	48eac199-516a-	PbldrOrdBooksDeltaRprt	Q
23.05.2018 15:37:25	23.05.2018 15:37:25	24	➔	6_0.prddlvr.XBID_Hour_Power.10YDE-ENBW-----N	48eac199-516a-	PbldrOrdBooksDeltaRprt	Q
23.05.2018 15:37:25	23.05.2018 15:37:25	22	➔	6_0.mbr.CENEX	48eac199-516a-	CashLmtDeltaRprt	Q
23.05.2018 15:37:25	23.05.2018 15:37:25	18	➔	6_0.tg.CENEXxxxxxxxxxxxx	48eac199-516a-	OrdExeRprt	Q
23.05.2018 15:37:25	23.05.2018 15:37:25	17	➔	m7.private.responseQueue.CXOWEG02.6ada4630-d2da-42b3-bd18-694f323f34dc	48eac199-516a-	AckResp	Q
23.05.2018 15:37:25	23.05.2018 15:37:25	0	➔	m7.request.management	48eac199-516a-	OrdEntry	Q

- You can as well use the Java sample code executable JAR file to see the rest of message properties (for broadcast messages, since this code does send any request but the login one).

2.4.2 Inquiry Requests

Q: What is the purpose of Inquiry Requests?

A1: Inquiry requests are used to ask questions to M7: what is the content of this order book? Which own orders do I have? What is the list of own trades executed of a certain time period? Etc.

Q: Are there any constraints to use these inquiry requests?

A1: Please refer to the M7 API presentation and to DFS180 to get more details about

- Dependencies between these requests (e.g. retrieve Contracts before getting responses with Contract IDs)
- The order in which they are supposed to be sent

A2: Please refer to the “Terms of Reference” document to see how your application should be using these requests. Basically, they should be used only when starting your application, to “initialize” it (build a market context) and then only in case a recovery procedure needs to get triggered (e.g. unexpected disconnection, technical issue, broadcast message gap).

A3: Please check the inquiry request rate limit mechanism in the documentation. The proper limit values per 60 seconds and 60 minutes are sent in the *System Info Response* message.

Please note that these limit values do not apply to management requests, meaning that you can of course send and manage as many orders as you need to per minute and hour, as long as you respect the OTR rules (Order-To-Trade ratio, see below).

These limits apply only to information retrieval.

2.4.3 Management Requests

Q: Order management requests can be done by using a contract ID or a functional key (product, delivery start and end time). What is the best approach?

A1: the possibility to use a “functional key” (functional description of the contract) is mainly offered for UDB (User Defined Blocks) contracts (e.g. 12-15_XB).

Indeed, such contracts are not generated in advance by M7, but only generated on the fly when the first order for it is received by M7.

When this happens, M7 broadcasts a *Contract Information Report* containing the new Contract ID.

For other situations, we recommend to use Contract IDs. This will provide you the necessary independence layer to avoid you any future maintenance work in case products are changed (new name, new products, etc.)

Q: Are there any limitations regarding the use of management requests?

A: There is no technical 60s or 60 mn limit like for inquiry requests, but you must respect the OTR rules (Order-To-Trade ratio, see below).

Q: Order entry – what is the meaning of the Account Type (A/P)?

A:

- A = Agent Trading : use this A value when entering an order for a client. Put the name of the client in the Text attribute. Indeed, there is no real “On behalf” functionality for this kind of “broker” situation.
- P = Proprietary trading : use P when trading for your own account (regular situation).

2.4.4 Revision Number

Q: Can I use the revisionNo to detect a missed broadcast?

A1: specific behavior for order RevisionNo for Remote Products:

- As stated in the DFS180 the order RevisionNo is increased, for a given order id, in case of a partial execution, hibernation, modification without execution priority change.
- In particular, be aware that for technical reasons:
 - a. For a Local order, RevisionNo:
 - i. starts at 1 at submission time, and then increases,
 - ii. restarts at 1 when an order operation leads to an order id change.
 - b. For a Remote order, RevisionNo:
 - i. will start at 2 instead of 1,
 - ii. does not restart at 1 when an operation leads to an order id change (ex: price modification) and can be discontinuous (ex: 3, 5 instead of 3, 4, 5).
- Thus, for both local and remote products:
 - Please never use this RevisionNo to find out if some order broadcasts have been lost, but use the dedicated “Sequence number” instead.
 - The only valid rule that prevails for the RevisionNo is that it can only increase for a given order id.

Q: How should I use the revisionNo?

A: The revision number is part of responses to inquiry requests and of broadcasts.

When starting/initializing your application, or after any recovery procedure where inquiry requests are needed, it can happen that:

- By the time your app sends the desired inquiry request, broadcasts start accumulating in the broadcast queue, containing data with a specific “functional key” (e.g. order book delta report for a given contract ID/area, each delta report having its own revisionNo)
- And once inquiry requests have been sent your app receives a response partially overlapping some broadcasts just received (please see the M7 API presentation slides “deduplicate” process).

This revision enables you not to process the same information twice, by discarding broadcasts messages which have already been processed in the response (or vice versa depending on your implementation design).

2.4.5 Broadcasts (including heartbeats)

Q: No M7 heartbeat received.

A1: Connection to the AMQP broker is not established or has been lost.

A2: If the connection to the AMQP broker is still active, the M7 service might not be connected due to maintenance.

A3: Check the properties in your message header: ensure contentType =6.0 (major version of the API schema 6.x).

- The value must be 6.0 even if you are using a more advanced 6.x API schema (e.g. 6.5).
- If sending anything else than 6.0, your API application will not even receive an User Report in response to a LoginReq. No broadcast would be received either and thus no heartbeat.

Q: I'm not able to start ComTrader.

A1: ComTrader needs a Java Plugin (Java Runtime Environment) with the version 8 update 60 or higher (the latest 64-bit version is recommended)

Q: Which schema version is used by M7 in broadcast routing keys?

A1: M7 API always values the schema version with 6_0, for all schemas 6.x (6.0 being the “major” version of the 6.x family). No update is required until the next major version 7_0, probably in 2020.

Q: how long do messages live in the Broadcast queue? When testing my application and interrupting it without closing the connection or sending a Logout request I get old messages when reconnecting again and consuming the broadcast queue.

A1: if an API app disconnect gracefully, i.e. by sending LogoutReq, the broadcast queue content is deleted by the server. If an app just disconnects, Rabbitmq deletes the queue content only after 6 minutes of inactivity. (Note : as of M7 6.11 this broadcast queue TTL Time To Live will be reduced to 3 minutes).

Q: When should I register a consumer to my API user broadcast queue?

A: We recommend to register a consumer as of login is done (reception of a User Report), before sending any other inquiry request (to avoid losing broadcasts that might have been stored during an unexpected AMQP disconnection period that would have resulted in an automatic broadcast queue consumer deletion)

2.5 Recovery procedures

2.5.1 AMQP connection or heartbeat loss

Q: What is the purpose of the AMQP heartbeat?

A: Please consult <https://www.rabbitmq.com/heartbeats.html>

Q: How do I detect an AMQP issue?

A1: The AMQP client library (e.g. Java) has a build-in functionality to detect **AMQP issues like connection loss / heartbeat loss** and so on. In order to process AMQP issues properly, a **shutdown listener** needs to be registered and implemented. This enables the capability to recover in a graceful way and if required to log and notify your alerting system.

Please find more details here <https://www.rabbitmq.com/api-guide.html#shutdown>

In particular:

A **ShutdownSignalException** can be caught and indicates if the exception occurred at the AMQP connection or channel level, and gives the reason why this exception was raised. This enables to distinguish between connection losses, heartbeat losses and different other potential root causes.

*“One can retrieve the ShutdownSignalException, which contains all the information available about the close reason, either by explicitly calling the getCloseReason() method or by using the cause parameter in the service(ShutdownSignalException cause) method of the **ShutdownListener** class.*

*The ShutdownSignalException class provides methods to analyze the reason of the shutdown. By calling the **isHardError()** method we get information whether it was a connection or a channel error, and **getReason()** returns information about the cause, in the form an AMQP method - either **AMQP.Channel.Close** or **AMQP.Connection.Close** (or null if the cause was some exception in the library, such as a network communication failure, in which case that exception can be retrieved with **getCause()**)”*

Note: by default, your AMQP library may trigger an **automatic recovery mechanism** that you need to prevent by closing explicitly the connection. Many customers suffered from this automatic reconnection after a **Change Password** request, leading to automatic reconnection attempts with the old password. After 5 wrong attempts the API user account would be revoked and it was no longer possible to connect without resetting the password (and thus calling market operations).

Q: What recovery procedure should my app trigger after an AMQP connection or heartbeat loss?

A:

1. Try to close all AMQP objects: private response queue, channels, AMQP connection
2. re-establish an AMQP connection + app start/login procedure described in the API presentation slides (including the “client failover concept” trying URL2 when URL1 is not available) , with an exponential back off mechanism, keeping in mind that the Login Inquiry request rate limit per hour (= 70) should not be hit
3. Back off example: wait for 10s before the 1st start, then 20s, 40s, then every minute

2.5.2 M7 Heartbeat loss

Q: After how many missed M7 heartbeats should I trigger a recovery procedure?

A1: After 3 missed heartbeats your app should consider M7 heartbeat has been lost.

Q: What recovery procedure should my app trigger once M7 heartbeat has been lost?

Q1: When our API app **misses 3 heartbeats in a row:**

1. **raise a functional alert** (email to your IT and operations team):
for instance if your operations are aware of an M7 maintenance window, they may be willing to stop your application and restart it as soon as Market Ops announce the end of the maintenance window
2. **Try to log out from M7**
3. continue with the same recovery procedure as for an AMQP connection or heartbeat loss (close all AMQP objects, etc.)

2.5.3 Message Overflow Handling : Unroutable request detected by the Return Listener

Q: We have registered to the return listeners, what shall be our action when the message is rejected by the AMQP server ?

A1: this means that M7 is down or too busy to be able process the request: the request cannot routed (if the request is sent with the "mandatory" flags set).

- When this happens, your app can resend the request with an exponential back-off mechanism, knowing that if M7 is down it should detect it within the next 15s or so (3 missed M7 heartbeats, sent every 5 seconds + a margin taking into account a potential latency). If these ~15s seconds are reached then apply the “AMQP connection loss” recovery procedure
- During the whole process, please make sure that if you send an inquiry request you pay attention to the inquiry request rate limit (per 60s and per hour), to ensure you do not decrease your quota too fast (please check the values in the System Info Response message).

2.5.4 Gap detection

Q: What is a message Gap?

A1: A gap is a discontinuity in the broadcast messages sequence, for a given routing key (family of broadcast messages).

As a result, depending on the concerned messages that were not received, your application:

- Missed new data:
 - this is the case of own and public trades: a new trade maybe have been missed
- Missed a data update / has an outdated vision of data:
 - for own and public trades: a new status update (recalled, cancelled) has been missed for an existing trade.
 - this is the case as well of order book messages. As soon as your app missed one order books delta report the content is inconsistent : which order is still valid?
 - For a contract it could be that your app has missing the opening or the closure of trading for a given delivery area.
 - Etc.
- Receives old data (see “ignore” below)
- Receives the signal that M7 has been restarted and that broadcasts sequences are reinitialized (see “Reset” below)

Q: Why do gaps happen in test or production environment?

A1: Gaps can happen because:

- Your application does not close the AMQP connection when receiving a Logout Report:
 - When your application is kicked out by another one using the same API user your application receives a Logout Report and should react to it:
 - **the logged/kicked out application should absolutely close the AMQP connection** so that it does not continue to consume the broadcast queue ; otherwise there would be 2 consumers stealing messages from each other, leading to an inconsistent set of messages (each app would see missing messages, resulting in Sequence ID gaps).
- One message has been lost (e.g. potential AMQP bug).
- The order of messages sharing the same routing key has been changed (e.g. 10,11, 13,12: please see the gap recovery procedure description below).
- Your application consumes broadcasts from different channels (should not happen): one is “stealing” messages from the other.
- Your application does not consume messages fast enough.
 - Messages in the queue have some Time To Live after which they are deleted (see 3.2.7 in DFS180). If the client application is not fast enough when reading the messages, some may be deleted and thus causing gaps.

Q: How often do gaps happen in production?

A1: Gaps do not happen frequently but do happen: it is strongly recommended to implement a gap-detection mechanism as described in DFS180 chapter 3.2.4 Sequence counting for Broadcast Messages to prevent data-loss.

A2: historically they happen when the system is under high stress. Most of the time broadcasts are not exactly missing but delayed (below 5 seconds). But if you use algorithms you probably have configured a lower time between 1 and 5 seconds to quickly detect any missing message (and react to any inconsistent order book state for instance by re-inquiring order books).

That is why **it is essential to implement a gap recovery procedure.**

A3: Our provider constantly improves performances and monitoring/auto-healing of M7 to minimize these gaps (when caused by M7/Rabbit MQ). But gaps can also be caused by Internet or customers networks.

Q: What should I do if I detect gaps in test or production?

A: Please contact us and send us the logs from your application AND try to reproduce the gap issue with ComTrader (see below): ideally with the same user or with another user not used in production. If you happen to have only one user please contact Market Operations to get a second Read Only user for that purpose.

Q: How does the M7 API enable to detect a gap?

A1: Please refer to:

- DFS180 chapter 3.2.4 Sequence counting for Broadcast Messages explains how to rely on a broadcast attribute called “Sequence number” to detect a gap per routing key.
- The M7 API presentation

Note: you should never rely on revision numbers to detect a gap. Revision numbers should always increase (except when the back end restarts : they are reset to 0) but not necessarily by 1 (e.g. 0,1,3,5,6 etc.).

Please note that M7 or RabbitMQ does not guarantee any sending or reception order of messages across several routing keys.

This means that your API app should not be designed based on a specific message order reception across several RK (e.g. Order Execution Report vs Public Order Books Delta report, Order Execution Report vs Trade Capture Report, etc.).

When you observe the M7 API behavior you will see a very common messages order:

- e.g. in case of an order entry of a buy order with a positive price without any matching:
 - Acknowledgement Response,
 - Then Order Execution Report,
 - Then Cash limit Delta Report,
 - Then Public Order Book Delta Report,
- but please keep in mind that under stress situations that reception order might change.

Q: Can I have a false positive in terms of gap detection? Can it be a bug on my app side?

A1: Yes we had a few customers experiencing the following bug:

- their application was not functionally processing all broadcasts sent by M7 (like most apps). For instance the app was only interested in trades messages.
- but they had forgotten to at least take into account the sequence number of such messages before discarding them, when the message Routing Key was also used by other messages

A2: the false gap detection can come as well from a problem with the broadcast queue consumer

A3: it can come as well from a wrong dependency expectation implemented in the app. For instance, if the app is only “configured” to handle a private trade broadcast (Trade Capture Report) before the corresponding public trade broadcast (Public Trade Confirmation Report), then the app may ignore a public trade message in the rare situations where it would arrive before (NB : the message order between broadcasts with different RK is NOT guaranteed by M7).

- some broadcasts can arrive **faster** or **later**, frequently enough to interfere with algorithmic trading in case of dependencies in your app :

Routing Key	Correlation ID	Request Type
6_0.prddlvr.XBID_Hour_Power.10YFR-RTE-----C	23465073-d573-4d9a-ad50-e30f28f9c668	PblcOrdBooksDeltaRprt
6_0.prddlvr.XBID_Hour_Power.10YBE-----2	23465073-d573-4d9a-ad50-e30f28f9c668	PblcOrdBooksDeltaRprt
6_0.prddlvr.XBID_Hour_Power.10YNL-----L	23465073-d573-4d9a-ad50-e30f28f9c668	PblcOrdBooksDeltaRprt
6_0.pblc.trd.XBID_Hour_Power	23465073-d573-4d9a-ad50-e30f28f9c668	PblcTradeConfRprt
6_0.mbr.TM001	23465073-d573-4d9a-ad50-e30f28f9c668	CashLmtDeltaRprt
6_0.hiftrd.11XTM1-TM-----1	23465073-d573-4d9a-ad50-e30f28f9c668	TradeCaptureRprt
6_0.mbr.TM001	23465073-d573-4d9a-ad50-e30f28f9c668	CashLmtDeltaRprt
6_0.mbr.TM001	23465073-d573-4d9a-ad50-e30f28f9c668	CashLmtDeltaRprt
6_0.bg.11XTM1-TM-----1	23465073-d573-4d9a-ad50-e30f28f9c668	OrdExeRprt
m7.private.responseQueue.CXSIM107.c7e7dd24-7e70-4dec-a28f-52	23465073-d573-4d9a-ad50-e30f28f9c668	AckResp
m7.request.management	23465073-d573-4d9a-ad50-e30f28f9c668	OrdEntry

Q: What kind of recovery procedure should I implement to recover from a gap/discontinuity?

A1: M7 API presentation diagrams illustrate different types of gaps:

- [new sequence nb for a given routing key] > [Last one + 1]: “Real gap”
- [new sequence nb for a given routing key] = 0 : “Reset”
- 0 < [new sequence nb for a given routing key] < [Last one + 1]: these gaps should be ignored, that is the broadcast should not be processed (no functional added value)

A2: For the a) and b) types of gaps the M7 API presentation shows that the recovery procedure is compound of:

- A gap recovery procedure: wait for maximum a couple of seconds to see if the missing broadcast is just a bit late,
- If not received in the given timeline, launch a re-inquiry procedure:
 - Only for the impacted Routing Key
 - For all RKs as soon as the new sequence number = 0:
 - this means that the M7 back-end has been restarted:
 - this means that all sequence numbers for all RKs are going to be reset to zero: it is immediately possible to re-inquire for all RKs.
- Note: many customers implement a re-inquiry for all RKs even if only one is impacted. This enables them to maintain only one single recovery procedure in all the different scenarios. The inquiry part is then the same as during the app initialization/starting phase.

A3: Regarding the re-inquiry part: you cannot request “what you missed”, for the following reasons:

- 1) M7 does not enable to request specific sequence IDs. It enables to request a functional scope using different criteria.
- 2) You cannot guess for which message the gap occurred when a routing key is used to distribute several messages (e.g. 6_0.prd.<prodName> in the table below):
- 3) Even if an RK is used by one single message, the gap is experienced at a level which is not the lowest level you can request:
 - for instance let's assume your app experiences a gap with RK = 6_0.prddlvr.XBID_Hour_Power.10YDE-RWENET---I
 - The only message to be distributed via this RK type 6_0.prddlvr.<prodName>.<dlvryAreaId> is PblcOrdBooksDeltaRprt.
 - The RK clearly shows which order books area is concerned by the gap.
 - But it does not enable to identify the contract for which the order book update broadcast has been missed.

- As a result your app needs to re-inquire the whole order book content for the whole product and area indicated in the RK for which the gap occurred.

When a RK is used by several broadcast messages, just re-inquire messages your application follows/processes.

The table below shows how to handle the most common gap situations:

Routing Key (RK)	Broadcasts using this RK	Request to re-inquire	Comment
6_0.prddlvr. <prodName>. <dlvryAreaId>	PblcOrdrBooksDeltaRprt	PblcOrdrBooksReq for <prodName> and <dlvryAreaId>	All order books for this product and area are returned. That is all obk for all <contract Id/area> pairs of this product. Only take into account broadcasts for each contract id.area of this product with a higher revision number than the one contained in the response to the inquiry request.
6_0.pblc.trd. <prodName>	PblcTradeConfRprt, sent when a trade is: <ul style="list-style-type: none"> • executed, • recalled • or cancelled by the operator 	PblcTradeConfReq for <prodName>. Adjust <i>startDate</i> and <i>endDate</i> to retrieve only the executed trades that can have been missed (*)	(*) <i>startDate</i> and <i>endDate</i> concern the trades executed during that period. But a trade can be executed at a given time and be recalled a few minutes later or be cancelled until the end of trading time. As a result <i>startDate</i> should be positioned at trading start time for the earliest contract trading time of the delivery areas your app follows.
6_0.bg.<acctId>	OrdrExeRprt	OrdrReq for the BG <acctId>	Caution: order request is done for all BGs when the Account Id is not provided.
6_0.hlftrd.<acctId>	TradeCaptureRprt	TradeCaptureReq for the BG <acctId>. Adjust <i>startDate</i> and <i>endDate</i> to retrieve only the executed trades that can have been missed (*)	
6_0.prdd.<prodName>	ProdInfoRprt	ProdInfoReq for <prodName>	Only if your app follows this message.
	ContractInfoRprt	ContractInfoReq For the .<prodName> product Adjust <i>startDate</i> and <i>endDate</i> to retrieve only contracts that may have changed (opened or closed) recently: contracts for delivery: <ul style="list-style-type: none"> • D+1 • and D since the timestamp of the last 	Your app necessarily follows contracts. Note: <i>startDate</i> and <i>endDate</i> refer both to contract delivery interval and to the date/time at which contracts are tradable.

		received message (with latency margin). This might be on D-1 if the gap occurs just around the switch from D-1 to D (D = current day)	
	RefPxRprt	RefPxReq for all contract IDs of <prodName> for opening prices (Type = "O")	Only if your app follows this message.
	MsgRprt	MsgReq for all Account IDs (balancing groups) assigned to the connected API user.	Only for applications having a GUI and displaying M7 messages. MsgReq does not allow to request messages only for a given product.

Q: Does ComTrader detect gaps?

A1: Yes, ComTrader does detect gaps, at 2 levels:

- in ComTrader logs:
 - o a new row is created when ComTrader detects a discontinuity in the Sequence ID of a Routing Key:
 - 2019-09-13T14:51:37.171+0200 [ader-Receiver-9] ERROR c.d.c.c.s.a.MessageSequenceCheck - Gap detected: 6_0.prddlvr.XBID_Hour_Power.10YFR-RTE-----C 448979>449121.
 - o At that moment ComTrader starts a “gap recovery mode”, that is starts waiting for 10 seconds for the missing broadcast, and traces this in its logs:
 - 2019-09-13T14:51:37.171+0200 [ader-Receiver-9] INFO c.d.c.c.s.a.SequenceCheckBroadcastListener - Starting gap recovery mode
- If the missing broadcast is not received after 10 seconds, ComTrader disconnects the user and display the gap issue in a pop-up

2.5.5 Inconsistent revisionNo

Q: What should I do when I receive (for the same functional broadcast message, i.e. for the same trade ID) **a Revision No that is “inconsistent”** (that is equal or lower to the previous one, once messages have been sorted by sequence number)

A:

- revisionNo are supposed to increase strictly, without necessarily being continuous (a jump e.g. from 2 to 4 can happen, but still increasing).
- In case you receive the same revision number for the same functional broadcast message (e.g. same trade ID, or order book for the same contract ID/area) twice in a row (after having sorted messages per sequence nb) : this is called a “technical” update:
 - o ignore the message only if it is a technical update, i.e. when all other characteristics but the revision No are identical (e.g. for a trade only the state can vary from one message to another, because a recall or cancellation process is going on)
- If equal (and not a “technical” update) or lower:
 - o This is an inconsistency that requires to trigger a recovery procedure (e.g. for public trades, send requests trades around the trade execution time indicated in the trade broadcast to get the latest version of trade)

2.6 Functional

2.6.1 Contracts

Q: What is the typical contract life cycle?

A1: All contracts but UDB (User Defined Blocks) follow the same life cycle:

- Creation: are generated several days in advance in M7, based on an M7 “schedule” which defines the list of possible phases (Active, Hibernated, Standby, Closed) for each delivery area.
- Trading start/end: at each contract/area phase state change, M7 API sends a Contract Information Report with all details.
- Closed: the contract is not tradable anymore

Note: Please always consider only area contract info (state, trading start/end) and NOT information featured at the contract level (technical info necessary to encompass all areas).

Q: what is the difference between the contract (short) name and long name?

A1: M7 features 2 contract names:

1. Contract **short name**: *name* attribute in the API, with a variable content:

- from contract creation until contract renaming:
At the contract creation, short name = long name.
The contract renaming occurs a day before delivery a midnight (CET/CEST)
This is not visible in ComTrader as only the (almost) tradable products are included. But is visible in the API *ContractInfoRprt* messages.
- from contract renaming until delivery day included:

In the 'Market overview' order book panels:

Market Overview (Predefined products)													
R	+	Area	Ctrct	TmZ	Cur	Phase	State	BVWA	BAcc	OBid	BQty	Bid	Ask
⊗	⊕	NGET	HH180914-21	Euro	GBP	CONT	ACTI	37.00	14.0		14.0	37.00	
⊗	⊕	NGET	HH180914-22	Euro	GBP	CONT	ACTI	31.00	12.0		12.0	31.00	35.00

In the 'Own trade' panel:

Own Trade		
TradeID	Ctrct	LCtrct
12308512	HH180914-21	20180914 10:00-20180914 10:30
12307924	HH180913-30	20180913 14:30-20180913 15:00

- after delivery day:
 - for Hourly and Half Hourly contracts : short name = long name
 - this is not visible in order book panels since only tradable contracts are displayed in those panels,
 - but this is visible in the "Own Trade" panel: the short name has been updated with the long name ("LCtrct" column)

Own Trade		
TradeID	Ctrct	LCtrct
12308512	20180914 10:00-20180914 10:30	20180914 10:00-20180914 10:30
12307924	20180913 14:30-20180913 15:00	20180913 14:30-20180913 15:00

- for contracts with a delivery period > 1 hour: short name remains unchanged (for historical reason related to continental markets)
- The short name pattern takes into account the product time zone (London time for GB products)

2. Contract **long name**:

- contains the delivery period in the product time zone (London time for GB products)
- does not change along the contract life cycle

Own Trade		
TradeID	Ctrct	LCtrct
12308512	HH180914-21	20180914 10:00-20180914 10:30
12307924	HH180913-30	20180913 14:30-20180913 15:00

Example with a Half Hour contract:

Contract name	In ComTrader	In API Contract messages	Content
Short Name	"Ctrct" column, displayed in: <ul style="list-style-type: none"> - Order book panels - Own Trade 	"name" attribute	<ul style="list-style-type: none"> • From 9 Sept. until 14 Sept: 20180914 10:00-20180914 10:30 • until 14 Sept.: HH180914-21 • as of 15 Sept: 20180914 10:00-20180914 10:30
Long Name	"LCtrct" column	"longName" attribute	Always: 20180914 10:00-20180914 10:30

Implementation recommendation

Caution:

- API wise we recommend identifying contracts by their *product + delivery start + delivery end* rather than by contract names.
- in the corresponding API ContractInfoRprt message Delivery Start and End are in UTC time format: *dlvryStart="2018-09-14T09:00:00.000Z" dlvryEnd="2018-09-14T09:30:00.000Z"*

Q: How often do I need to retrieve Contracts for my products? Once a day?

A: You only need to send a Contract Information Request when your application starts or after a recovery procedure. Then while connected and logged in the new contract IDs will be communicated via (Contract Info Response) broadcasts.

Note1: your app must manage a Contract referential (e.g. table in memory, database) containing all contract IDs and related info (per area: state, phase, trading start/end) from their creation until the moment where you app does not need them anymore (depending on how long your app needs historical data after contract closure).

Note2: With such a mechanism in place, your app should never be in the situation where it receives an unknown contract ID in another Message (e.g. *Public Order Book Delta Report* or *Public Trade Confirmation Report*).

Q: How can I know that a contract is tradable?

A: You need to rely on Contract phase and state per area. Please refer to the M7 API presentation slides :” M7 Contract life cycle and messages” and “Contract and Phase States”

Q: How can I get from the API the list of phases that a contract will go through?

A: It is not possible to get this information via the API. Indeed, *Contract Information Reports* only display the start and end date/time of the current phase.

Please refer to the dedicated Excel Sheet in the M7 API Package to know the content of the entire contract schedule (phases sequence).

Q: I see 2 states in Contract Information Report messages. Which one should I consider?

A: The state at the contract level should be ignored since it is a technical state.

Please consider only the state of each area phase.

Please see as well “How can I know that a contract is tradable”.

Q: where can I find DST (Daylight Saving Time / clock change) contract information?

A: Please consult DFS130 M7 DST Behavior.

2.6.2 Order books

Q: Which logic should be followed to process Public Order Books messages

A1: Your API application must:

- first send an inquiry request (*Public Order Books Request*) to get full order books content
- and then listen to order book updates (*Public Order Books Delta Report* broadcasts). Send again an inquiry request only if required (e.g. disconnection, discontinuity/gap in broadcasts sequence number in the order book routing key, M7 heartbeat loss)
 - Your API user receives all updates for the products and delivery areas it is assigned to (via its BG/account)

Note: Order book messages refer to contract IDs : you app must send *Contract Information Requests* when it starts (or when it recovers from any event, see recovery procedures), and then update Contract information by processing continuously *Contract Information Reports*.

With such a mechanism your app should never be in the situation where it receives an unknown Contract ID and send a Contract Information Request for that contract ID. You can implement such a behavior as a safety measure. But this should trigger a WARNING so you can investigate and fix the root cause.

Q: What can happen to an order when submitted by an API application?

A1: Orders can be sent into the market with an initial hibernated or active state.

- **Hibernated (deactivated) orders** are not featured in the order book (obk). M7 enables to send orders with an initial hibernated state. Traders/app activate them later on when required.
- **Active orders:** all submitted active orders do not necessarily get in the order book:
 - Only the unmatched part of an order goes into the obk if its execution restriction in NON**
 - If the order execution restriction is IOC “Immediate Or Cancel” then the unmatched part does not go into the obk (deleted by the system)
 - If an order gets fully matched then a trade is executed but there is no trace of this order in the obk ; the order book will only reflect the reduction of quantity of the order(s) that got matched (and the executed trade in the Last Trade Info).

Example:

Let's consider an order book with 2 passive orders that get matched at the same time by an aggressor order, the remaining qty of the aggressor being inserted in the order book:

- submit in AMP a buy order 5MW@10€

```
<?xml version="1.0" encoding="UTF-8"?>
<PblcOrdrBooksDeltaRprt xmlns="http://www.deutsche-boerse.com/m7/v6">
  <StandardHeader marketId="EPEX"/>
  <OrdrbookList>
    <OrdrBook contractId="66889" dlvrAreaId="10YDE-RWENET---I" lastPx="1000"
      lastQty="5000" totalQty="110000" lastTradeTime="2019-04-05T14:32:49.520Z" pxDir="-1"
      revisionNo="8" highPx="2400" lowPx="1000">
      <BuyOrdrList>
        <OrdrBookEntry ordId="116177323" qty="5000" px="1000" ordEntryTime="2019-04-
          05T14:46:17.842Z"/>
      </BuyOrdrList>
    </OrdrBook>
  </OrdrbookList>
```

```
</PblcOrdrBooksDeltaRprt>
```

- submit in AMP a buy order 4MW@9€ (same contract)

```
<?xml version="1.0" encoding="UTF-8"?>
<PblcOrdrBooksDeltaRprt xmlns="http://www.deutsche-boerse.com/m7/v6">
  <StandardHeader marketId="EPEX"/>
  <OrdrbookList>
    <OrdrBook contractId="66889" dlrvyAreaId="10YDE-RWENET---I" lastPx="1000"
      lastQty="5000" totalQty="110000" lastTradeTime="2019-04-05T14:32:49.520Z" pxDir="-1"
      revisionNo="9" highPx="2400" lowPx="1000">
      <BuyOrdrList>
        <OrdrBookEntry ordId="116177324" qty="4000" px="900" ordEntryTime="2019-04-
          05T14:47:00.616Z"/>
      </BuyOrdrList>
    </OrdrBook>
  </OrdrbookList>
</PblcOrdrBooksDeltaRprt>
```

- Submit in AMP a sell order 10MW@8€ (same contract): this order gets matched against the 2 passive ones (already in obk). The remaining part gets inserted in the obk.

- Please note that the Obk message can arrive before or after the Public trade one, you should not rely on any sequence order.

```
<?xml version="1.0" encoding="UTF-8"?>
<PblcTradeConfRprt xmlns="http://www.deutsche-boerse.com/m7/v6">
  <StandardHeader marketId="EPEX"/>
  <TradeList>
    <PblcTradeConf tradeId="13159898" state="ACTI" contractId="66889" qty="5000"
      px="1000" tradeExecTime="2019-04-05T14:47:51.734Z" revisionNo="1"
      buyDlrvyAreaId="10YDE-RWENET---I" sellDlrvyAreaId="10YDE-RWENET---I"/>
    <PblcTradeConf tradeId="13159899" state="ACTI" contractId="66889" qty="4000"
      px="900" tradeExecTime="2019-04-05T14:47:51.734Z" revisionNo="1"
      buyDlrvyAreaId="10YDE-RWENET---I" sellDlrvyAreaId="10YDE-RWENET---I"/>
  </TradeList>
</PblcTradeConfRprt>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<PblcOrdrBooksDeltaRprt xmlns="http://www.deutsche-boerse.com/m7/v6">
  <StandardHeader marketId="EPEX"/>
  <OrdrbookList>
    <OrdrBook contractId="66889" dlrvyAreaId="10YDE-RWENET---I" lastPx="900"
      lastQty="4000" totalQty="128000" lastTradeTime="2019-04-05T14:47:51.734Z" pxDir="-1"
      revisionNo="10" highPx="2400" lowPx="900">
      <SellOrdrList>
        <OrdrBookEntry ordId="116177325" qty="1000" px="800" ordEntryTime="2019-04-
          05T14:47:51.734Z"/> => remaining part of the aggressor order that got inserted in the order
        book. If this order would not have been inserted it would have been impossible to get its order
        ID.
      </SellOrdrList>
      <BuyOrdrList>
        <OrdrBookEntry ordId="116177323" qty="0" px="1000" ordEntryTime="2019-04-
          05T14:46:17.842Z"/> => qty = 0: the order got Traded
        <OrdrBookEntry ordId="116177324" qty="0" px="900" ordEntryTime="2019-04-
          05T14:47:00.616Z"/> => qty = 0: the order got Traded
      </BuyOrdrList>
    </OrdrBook>
  </OrdrbookList>
</PblcOrdrBooksDeltaRprt>
```

- **Special case of iceberg orders:**

- Order books only feature the shown (displayed) qty of an ICB order. The total quantity is hidden.
- If the ICB order gets traded up to the point where a new peak gets generated and that new peak gets partially traded, then you get only 1 order book update but 2 public trades:
 - i. let's take for instance a 50MW total qty order with a 10MW peak: initially the obk shows 10MW

- ii. If it gets aggressed by an 11MW order this results in 2 trades at the same price with qty 1 = 10MW and qty 2 = 1 MW

```
<?xml version="1.0" encoding="UTF-8"?>
<PblcTradeConfRprt xmlns="http://www.deutsche-boerse.com/m7/v6">
  <StandardHeader marketId="EPEX"/>
  <TradeList>
    <PblcTradeConf tradeId="13163878" state="ACTI" contractId="68288" qty="10000"
    px="1100" tradeExecTime="2019-04-08T11:49:46.122Z" revisionNo="1"
    buyDlvryAreaId="10YDE-RWENET---I" sellDlvryAreaId="10YDE-RWENET---I"/>
    <PblcTradeConf tradeId="13163879" state="ACTI" contractId="68288" qty="1000"
    px="1100" tradeExecTime="2019-04-08T11:49:46.122Z" revisionNo="1"
    buyDlvryAreaId="10YDE-RWENET---I" sellDlvryAreaId="10YDE-RWENET---I"/>
  </TradeList>
</PblcTradeConfRprt>
```

- iii. But you get one single Public Order Books Delta Report message where the (visible) Qty switches from 10MW to 9MW

```
<?xml version="1.0" encoding="UTF-8"?>
<PblcOrdrBooksDeltaRprt xmlns="http://www.deutsche-boerse.com/m7/v6">
  <StandardHeader marketId="EPEX"/>
  <OrdrbookList>
    <OrdrBook contractId="68288" dlvryAreaId="10YDE-RWENET---I" lastPx="1100"
    lastQty="1000" totalQty="22000" lastTradeTime="2019-04-08T11:49:46.122Z" pxDir="0"
    revisionNo="3" highPx="1100" lowPx="1100">
      <BuyOrdrList>
        <OrdrBookEntry ordId="116200681" qty="9000" px="1100" ordEntryTime="2019-04-
        08T11:49:46.122Z"/>
      </BuyOrdrList>
    </OrdrBook>
  </OrdrbookList>
</PblcOrdrBooksDeltaRprt>
```

- **Other events in the order book:**

- Any order deactivation or deletion will be translated in the obk as a “qty = 0” record
- M7 deletes orders when a contract closes
- M7 deactivates all orders when the market gets halted.

- **Order IDs logic / impact on order book:**

- There are a few order events where M7 cancels the existing order ID and submits a new Order having a new order ID:
 - For an active order:
 - Price change
 - Quantity increase
 - The order execution restriction is changed (NON -> IOC or NON->FOK)
 - For a hibernated order:
 - Activation
 - Type change between REG and ICB: though such an order would not appear in the obk since hibernated

- For instance, if a public order price (order ID = 115848168) is changed you would get the following message:
 - One record with the old order ID and qty = 0
 - One record with the new Order ID and updated price/qty and a new order Entry Time:

```
<?xml version="1.0" encoding="UTF-8"?>
<PblcOrdrBooksDeltaRprt xmlns="http://www.deutsche-boerse.com/m7/v6">
  <StandardHeader marketId="EPEX"/>
  <OrdrbookList>
    <OrdrBook contractId="56620" dlvryAreaId="10YNL-----L" lastPx="8256"
    lastQty="1000" totalQty="491800" lastTradeTime="2019-03-14T09:57:08.496Z" pxDir="0"
    revisionNo="149" highPx="8256" lowPx="-5050">
      <BuyOrdrList>
        <OrdrBookEntry ordId="115848168" qty="10000" px="3800" ordEntryTime="2019-03-
        14T09:58:20.453Z"/>
      </BuyOrdrList>
    </OrdrBook>
  </OrdrbookList>
</PblcOrdrBooksDeltaRprt>
```

```
<OrdrBookEntry ordId="115848168" qty="0" px="3900" ordEntryTime="2019-03-14T09:58:13.282Z"/> => old order being removed automatically by M7, qty = 0
</BuyOrdrList>
</OrdrBook>
</OrdrbookList>
</PblcOrdrBooksDeltaRprt>
```

Q: can I track the complete life cycle of a given order by analyzing Order book messages?

A: No there are situations where you would lose track of an order ID, e.g. if a market participant application modifies several orders at once.

Example:

- Enter 2 orders with text “Order 1” and “Order 2”.
 - Order 1 ID = 116100005
 - Order 2 ID = 116100007.
- Modify the price (or increase the quantity) of these 2 orders at the same time:

```
<?xml version="1.0" encoding="UTF-8"?>
<OrdrModify xmlns="http://www.deutsche-boerse.com/m7/v6" ordModType="MODI">
  <StandardHeader marketId="EPEX"/>
  <OrdrList>
    <Ordr ordId="116100005" px="3000" txt="Order 1"...>
    <Ordr ordId="116100007" px="3100" txt="Order 2"...>
  </OrdrList>
</OrdrModify>
```

- You get the following Public order book message:

```
<?xml version="1.0" encoding="UTF-8"?>
<PblcOrdrBooksDeltaRprt xmlns="http://www.deutsche-boerse.com/m7/v6">
  <StandardHeader marketId="EPEX"/>
  <OrdrbookList>
    <OrdrBook contractId="65427" dlrvyAreaId="10YDE-RWENET---I" lastPx="1100" lastQty="2000"
    totalQty="38000" lastTradeTime="2019-04-02T17:53:20.125Z" pxDir="-1" revisionNo="26" highPx="3700"
    lowPx="1100">
      <BuyOrdrList>
        <OrdrBookEntry ordId="116100011" qty="2000" px="3000" ordEntryTime="2019-04-02T17:54:42.109Z"/> => new Order 1
        <OrdrBookEntry ordId="116100012" qty="4000" px="3100" ordEntryTime="2019-04-02T17:54:42.109Z"/> => new Order 2
        <OrdrBookEntry ordId="116100005" qty="0" px="1000" ordEntryTime="2019-04-02T17:54:00.709Z"/> => Old Order 1
        <OrdrBookEntry ordId="116100007" qty="0" px="900" ordEntryTime="2019-04-02T17:54:17.045Z"/> => Old Order 2
      </BuyOrdrList>
    </OrdrBook>
  </OrdrbookList>
```

If you were not aware of the modify request content you might be tempted to conclude that 116100005 transformed into 116100011 for instance.

But actually when sending a modify request for several orders, modifications are applied one by one based on requests order rather than based on order IDs:

- the above assumption is correct only if the modify order starts with order 1.
- If it were starting by order 2 than 116100007 would have transformed into 116100011:

```
<?xml version="1.0" encoding="UTF-8"?>
<OrdrModify xmlns="http://www.deutsche-boerse.com/m7/v6" ordModType="MODI">
  <StandardHeader marketId="EPEX"/>
  <OrdrList>
    <Ordr ordId="116100007" px="3100" txt="Order 2"...>
    <Ordr ordId="116100005" px="3000" txt="Order 1"...>
  </OrdrList>
</OrdrModify>
```

As a result it is not possible in that situation to keep on following the “story” of Order 1 and 2, because of that uncertainty to establish this parent-child relation.

2.6.3 Orders

Q: What are orders possible lifecycle? What are the possible order states and possible transitions between these states?

A: Please refer to the M7 API Presentation Order State diagram.

Q: What are the order management actions that lead to a new order ID?

A1: several actions trigger a new Order ID, M7 first deleting/cancelling the order and then submitting it again (D+A).

- This is the case when:
 - The price of an active order is modified
 - The quantity (or Peak qty for icebergs) of an active order is increased,
 - The order execution restriction is changed (NON -> IOC or NON->FOK)
 - A deactivated order is activated
 - The type of a deactivated order is changed: REG <-> ICB

2.6.4 Trades

Q: What is trades lifecycle?

A: The trade lifecycle does not necessarily stop at the execution stage (ACTI state) but can as well be recalled (on request of traders/API users) but as well cancelled by the operator. Please refer to the documentation and check the Trade State diagram.

Q: I do not see the buy and sell order ID in a *Public Trade Confirmation Report*. How can I determine them?

A: Indeed the Public Trade Confirmation Report message does not offer the order IDs of the buy and sell orders that got matched into a trade. There is no way from a customer perspective to build it consistently in all situations.

2.6.5 Balancing Groups

Q: What is the meaning and role of Balancing Groups in the M7 system?

A1: A Balancing Group (BG) is an M7 entity that represents a **Unit**.

- Each BG is assigned to all SEMOpX products in the unit’s area.
- Each API user is assigned to all available BGs, unless otherwise specified by the member.
- One BG (Unit) can be linked only to one single M7 member entity.
- Please refer to the M7 API presentation to see diagrams illustrating these links.

Q: How can I check the BGs my API user is assigned to?

A1: You can check in the User Report (sent as a response to a Login Request) to which BGs (Units) your user is assigned to.

The resulting BGs, products and areas condition the scope of requests that can be sent and of broadcasts that will be received (based on the “Routing Key” mechanism).

A2: This can as well be checked in ComTrader order entry panel, by opening the “Balancing Group” dropdown list.

2.6.6 Indexes and VWAP

Q: Does the M7 API calculate any index or VWAP?

A: No, the M7 API does not calculate any of those.

2.6.7 Self-trades and Cross trades

Q: What is a self-trade or a cross-trade?

Self-trades are defined as trades where both counterparties of a trade (buyer and seller) are from the same exchange member (same ECC short name). This includes trades between two different delivery areas (= cross border self trades).

Public (PblcTradeConfRprt) and private (TradeCaptureRprt) trade messages feature the true/false selfTrade attribute to inform you a posteriori that the trade executed was a self-trade.

There is a subtle difference between self-trades and cross-trades, though we tend to use both terms indifferently.

The **cross-trade** concept was introduced with ComTrader (cross-trade protection setting) and can be extended to any API application: only the Balancing Groups assigned to the connected API user can be taken into consideration, when self-trades consider all BGs of the member indifferently.

Indeed, the objective of a cross trade protection, as explained below, is for a logged in API user to identify in Public Order Books which Order IDs belong to one of his assigned BGs, to avoid matching them.

And since because of the routing key mechanism an API app can only receive Order Execution Reports for the BGs its logged in user is assigned to, these assignments do make a difference: an API user cannot identify as an “own” order an order linked to a BG (Unit) on which it does not have any assignment/visibility.

As a result, it is possible to have a self-trade without having the possibility to prevent any cross-trade, if the API app user is not assigned to all BGs used by a member to place orders.

Q: Can I trade with ComTrader and have an API application at the same time? What is the cross-trade risk?

A1: the risk of a casual cross trade exists but is limited to very simultaneous actions done by the API apps (robot) AND the trader: they should be very occasional if all alerts are set up/properly implemented, as explained below:

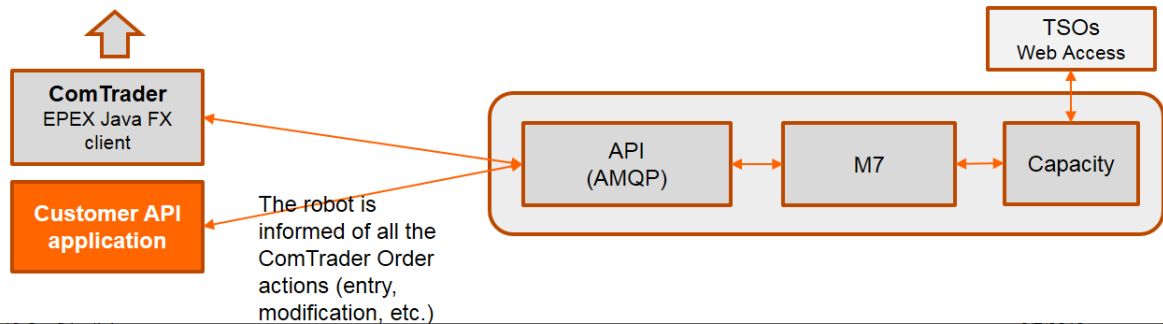
How to avoid cross trades between an API robot and a ComTrader user in most situations:

ComTrader enables traders to identify via a color code:

- their **personal** orders (below in light **Blue**)
- their **colleagues** orders (users belonging to the same BG): orders submitted by the API robot will belong to that category => the Trader can identify robot orders in order books
- market orders (in white)

► Market Overview (Predefined products)

R	+	Area	Ctrlct	TmZ	Cur	Phase	State	BVWA	BTQty	BAcc	OBid	BQty	Bid	Ask	AQty	OAsk	AVWA	AAcc	ATQty
✕		AMP	18-19_XB	CET	EUR	CLSD	ACTI												
✕		AMP	19-20_XB	CET	EUR	CONT	ACTI	20.03	5.0	5.0	5.0	5.0	20.03	20.99	9.8		20.99	9.8	9.8
								20.01	24.0	29.0		24.0	20.01	21.55	25.0		21.39	34.8	25.0
								19.68	4.0	33.0		4.0	17.30	23.00	5.0		21.50	39.8	5.0
								18.85	2.0	35.0		2.0	5.00	23.00	10.0	10.0	21.88	49.8	10.0
								15.55	10.0	45.0		10.0	4.00						



From a trader's perspective: ComTrader offers a “Cross trade alert” setting that needs to be activated in the trader user profile:

- **Case of a trader aggressor order:** if the cross trade alert is activated, in case the trader enters, activates or modify an order (aggressor order) that would match a robot order, **ComTrader should detect it in more than 99% of situations**, resulting in a **pop-up asking the trader to confirm** his order management action
 - Note: the alert will consider only the connected user orders, ignoring any market orders that could be in between, because they could disappear at any time. In other words, market orders cannot be considered as a protection.
 - Example: on the order book above, the sell order in white would be ignored: modifying the yellow order with a price of 23€ would trigger the cross-trade alert
- **Case of a passive order:** there is no cross-trade alert mechanism for passive orders (passive = orders already in the order book that would get traded by the robot: see “from the robot perspective”).
- **Cases where the alert would not work:**
 - **simultaneous actions by the robot AND the trader:** since this alert is implemented on the ComTrader side (that is not on the M7 back end side) there is still a risk that ComTrader is not yet aware of the robot order last order change when the trader sends the aggressor order, and therefore unable to detect a cross trade: the aggressor would match the robot order (which is not where ComTrader thought it was)

From the API robot's perspective:

- **Customers must implement a cross trade prevention mechanism**, with the same risks as mentioned for the ComTrader cross trade alert:
- Since the API user and the Trader user belong to the same BG, the API robot will receive messages (order execution report) for any trader order management action in ComTrader: the API robot algorithm will have all the information to avoid a cross trade.