NASDAQ OMX®

Genium INET℠

OMnet Message Reference Introduction

HKEX

Version: 2.0.0801

Document ID:	OMnet_MessRef_Intro_12
Documentation Release:	GENIUM_Product_a682
Release Date:	2013-01-09
Publication Date:	2013-01-09

Table of Contents

1 Introduction 5

2 Overview of the System and the API 7

2.1 Genium INET Trading 8
2.1.1 The Matching Engine 8
2.2 Genium INET Clearing 8
2.2.1 How Genium INET Clearing is Built Up 9
2.2.2 Multilateral Clearing 9
2.2.3 Member and Client Clearing 9
2.2.4 Linked Clearing 10
2.2.5 Genium INET℠ Clearing and the API 10

3 Description of the API 11

3.1 Login and Logout Transactions 11
3.2 Common Record Structures and Facilities 11
3.2.1 Basic Concepts 11
3.2.2 Transaction Headers 14
3.3 Partition Handling 15
3.3.1 Partitioning in Genium INET Clearing 15
3.3.2 Partitions on EP0, EP3 and EP5 16
3.4 General Query Transaction Handling 17
3.4.1 General 17
3.4.2 Segmented and Coupled Queries 17
3.5 Segmented Broadcasts Handling 18
3.6 Order Handling when the Remote System Terminates 18
3.6.1 Logout Timer 19
3.6.2 Inactivated Orders 19
3.7 Error Handling 19
3.8 Sequences 19
3.9 Trading and Instrument Sessions 20
3.9.1 Trading Session 20
3.9.2 Instrument Session 22
3.9.3 Active State 22
3.9.4 Variable Information Messages 25
3.10 The Virtual Underlying Concept 28
3.11 Clearing 29
3.11.1 Real time Information 29
3.11.2 Batch Information 29
3.11.3 Signals 30
3.12 Deal and Trade Numbers 30
3.12.1 Identification of Exercises 31

3.13 Order Transactions 31
3.13.1 Order Number 32
3.13.2 Order Transactions and Queries with Trader ID 32

List of Figures

Figure 1: Overview 7
Figure 2: The Matching Engine 8
Figure 3: Accessing the Clearing System 10
Figure 4: Series Definition 14
Figure 5: Schematic Overview of Where to Define the Trading Sesson and the Instrument Session 20
Figure 6: Trading States on Different Levels 21

1 Introduction

This document describes the interaction between an application and the Genium INET Trading and Genium INET Clearing systems.

This manual is one of several publications covering the OMnet API, and by using these you will be able to link your application to the central system functions.

Manual	How to use it
OMnet Application Programmer's Interface Manual	Explains how the OMnet API is built up. This is the first manual to read when you are learning how to program towards the OMnet API.
OMnet Message Reference, Introduction	The manual you are currently reading. It contains further information on how the API is built up and the concepts that are used. **Note:** This manual contains generic examples of transactions, broadcasts and queries that might not be applicable to your exchange configuration.
OMnet Message Reference	Here are all the transactions, broadcasts and queries that are available and used for a particular Exchange. This chapter starts with an update on what is new from the latest release. This is to help you find the news as fast and easy as possible.
System Error Messages Reference	Lists all the error messages you can possibly get while programming against the API. For every error message there is an explanation on what to do.

Other deliverables connected to the API	How to use it
The API kit	The API kit contents of • omnet.h files, the c definitions of the transactions • sample program.c, example program

2 Overview of the System and the API

The Genium INET Trading and Genium INET℠ Clearing systems are fully integrated electronic systems designed for fast and reliable trading of financial products.

From a design point of view, the integrated system is divided into four core central subsystems, one common database (CBD) and one communication system (OMnet).

The four core central subsystems are:

- Matching Engine (ME)
- Information Dissemination (IN)
- Clearing (CL)
- Settlement (SE)

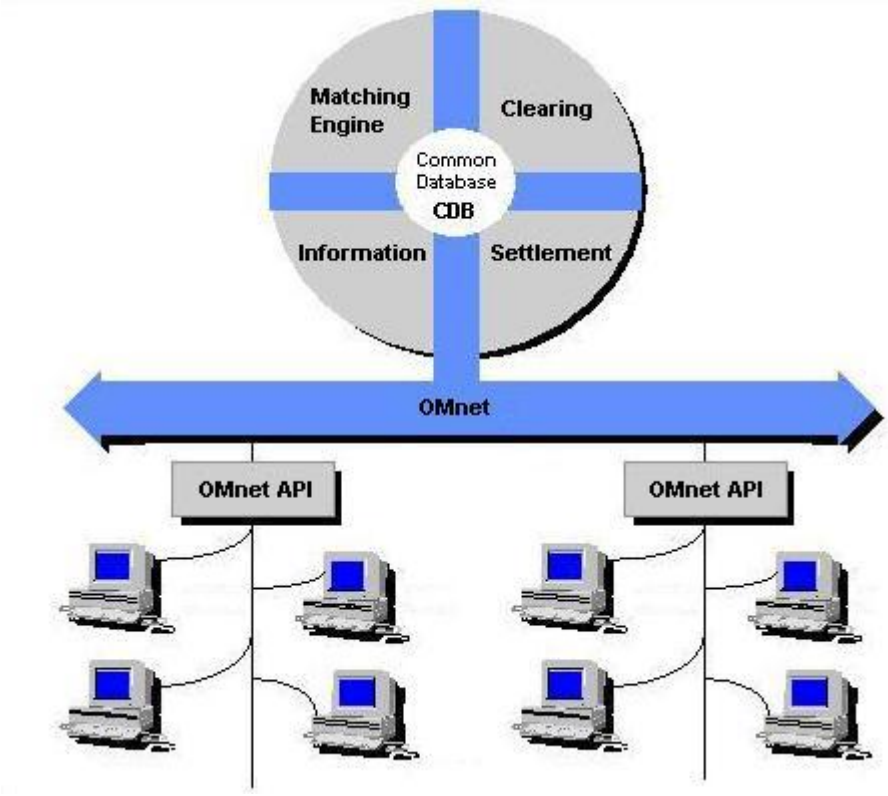

Figure 1: Overview

The system software is based on a client-server concept. The backbone of the system is a communication system called OMnet, which provides the members with an Open Application Programming Interface (API). The API gives the members direct interaction with the central software.

Members can interact with NASDAQ OMX supplied software or through their own developed trading applications or through commercially available trading applications.

2.1 Genium INET Trading

2.1.1 The Matching Engine

The Matching Engine's main task is to:

- Collect the bid (buy orders) and ask (sell orders) requests made for a particular instrument.
- Rank the orders and keep them in an Order Book.
- Execute matching orders and thereby closing deals.

In order to maintain a rapid electronic Matching Engine, it is necessary to have efficient Information Dissemination. The level of availability is a decision of the Exchange.

All remote functions will be connected to the Matching Engine via the OMnet interface. The design and manufacture of such functions will be left to the participants or third-party software suppliers.

The picture below shows how users from Remote Systems via the OMnet application programmer's interface can access the Matching Engine (including the Information Dissemination):

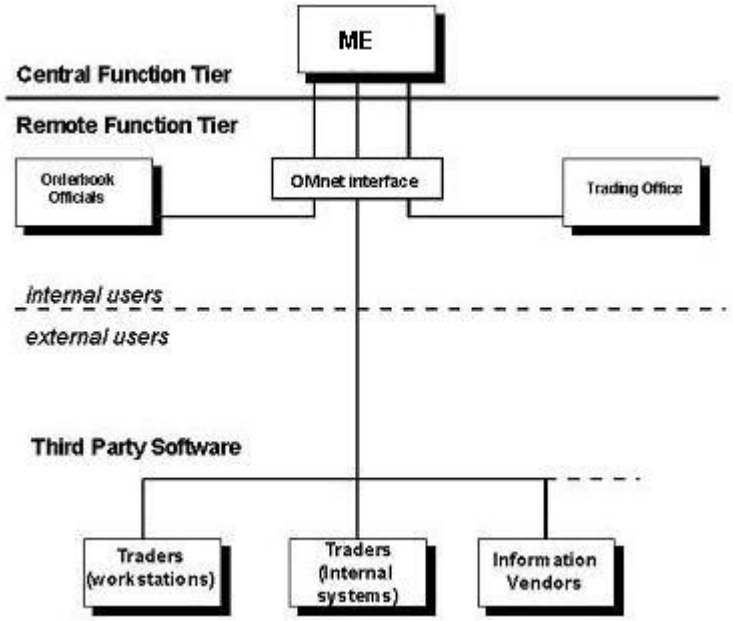

Figure 2: The Matching Engine

2.2 Genium INET Clearing

The main task for Genium INET℠ Clearing is to control and administrate deals and the clearing events. This includes all necessary functions after a trade has been made, for example:

- control and administration of trades and deliveries
- control and calculation of various kinds of settlements and margins

2.2.1 How Genium INET Clearing is Built Up

Genium INET℠ Clearing has been designed to ensure that a flexible clearing environment is created. A key part of the flexibility is obtained via the product and account model design.

- Product Model

  An event is a trigger that results in a process, the action. All the potential events and actions that can take place during a product's lifetime must be predefined in the system database (the Common Database, CDB). This gives flexibility to handle a variety of product types. Clearinghouse operators set up all anticipated events and actions when launching a product but can make amendments to the database if required at any time during that product's life.
- Account Model
- Genium INET℠ Clearing can co-operate with other clearinghouses in the clearing of products traded on a common exchange via the linked clearing API. For members who want to keep their connection to the exchange (not available at all exchanges), a back office function with clearing functionality is available.

Functions only obtained via OMnet are included in the OMnet Reference manuals. However, interest rate and currency functions are not completely covered.

2.2.2 Multilateral Clearing

Genium INET℠ Clearing is a multilateral clearing system designed to clear trades that are novated by a central clearinghouse (i.e. for which the clearinghouse takes over obligations). A single deal between a buyer and a seller is logically split into two or more trades each with the clearinghouse as a counterpart. The clearinghouse becomes both:

1. the seller to the buyer
2. the buyer to the seller.

2.2.3 Member and Client Clearing

Genium INET℠ Clearing can handle either member clearing or client clearing.

In a member clearing set-up a relatively low number of accounts exist, for example:

1. Principal accounts (own trades)
2. Agent accounts (all client trades)
3. Market Maker accounts (market maker trades)

In a client clearing set up a greater number of accounts are required, with one account (or more) for each client. The most common account types are:

1. Principal accounts
2. Daily Transitory accounts (client trades before allocated to a certain client)
3. Client accounts (one account for each client)
4. Market Maker accounts

A client (or member) must have a trading membership at the exchange but does not necessarily need a clearing membership. The following memberships are possible:

1. Direct clearing member, DCM, (member with both trading and clearing membership)
2. Non-clearing member, NCM, (trading but not clearing membership, clearing must be arranged via a GCM)
3. General clearing member, GCM, (member can clear both own and other members' trades)

2.2.4 Linked Clearing

Genium INET℠ Clearing can handle linked clearing where the members of an exchange clear at more than one clearinghouse and/or the participants of the local Genium INET℠ Clearing clearinghouse are members of more than one exchange.

The system handles clearing for its participants regardless of at which exchange the participant traded at. Participants that clear at external clearinghouses are anonymous to the local clearinghouse and their positions are aggregated into one summary account per external clearinghouse.

External clearinghouses use the OMnet API to exchange information with the local clearinghouse for exercise, position updates etc.

2.2.5 Genium INET℠ Clearing and the API

The following figure shows how users from remote systems via the OMnet application programmer's interface can access the clearing system.

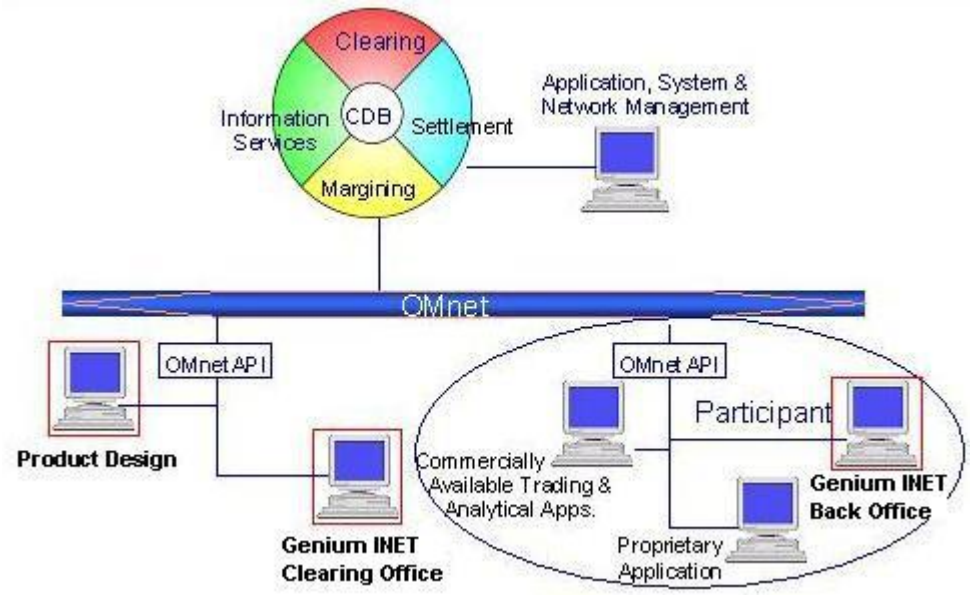

Figure 3: Accessing the Clearing System

3 Description of the API

3.1 Login and Logout Transactions

To log in to the system, a login transaction must be sent containing user name and password. After the Central System has made an authorization check, it is possible for the user to send the authorized transactions.

A login transaction is the start of a new session. If a new session is started without exiting the first one by a logout transaction, the Central System automatically creates a logout for any older session. Therefore a user cannot login with the same user name from more than one location at a time.

For details regarding all session-related transactions, see section *"Facilities and Facility Types"* in *OMnet Application Programmer's Interface Manual.*

3.2 Common Record Structures and Facilities

This section describes the basic data that is used in every transaction. This basic data is needed by the system to know which the responsible server is, and how to interpret the transaction.

3.2.1 Basic Concepts

3.2.1.1 Facilities

Transactions are sent on various paths, called *facilities*. The system will only be able to interpret the transaction if it is sent on the correct facility. Therefore, each transaction definition in the message reference manuals contains the facility on which the transaction must be sent. The facility is an argument to some of the OMnet API calls.

Some OMnet API calls take *facility type* as an argument. In general, facilities describe transaction paths. For instance, Order Transactions and Deal Capture Queries use different paths. In order to specify target facilities for transactions and queries, *facility types* are used.

The facilities names hold information about Country, Target Platform and Production Status.

> *Example 1:*
>
> At the NASDAQ OMX Exchange in Stockholm the facility **OMX__XSEEP0** is used for Order Transactions in the Production System and **OMX__XSEET0** in the Test System. If this document refers to facility EP0, the facility name is implicitly **OMX__XSEEP0** for NASDAQ OMX Stockholm Exchange Production Order Transactions and **OMX__XSEET0** for Test Transactions. If intending to access the EP0 test facility at the WBAG Exchange, use the facility type literal OMNI_FACTYP_OTOBXATET0 in the OMnet API calls.
>
> For a complete description of *facilities* and how to get the corresponding numbers, please refer to *"Facilities and Facility Types" in OMnet Application Programmer's Interface Manual.*

3.2.1.2 Transaction Type

The *transaction type* makes it possible for the system to identify the message, thus making it possible to interpret the transaction content. Consequently, the transaction type must be completed for all transactions.

The Transaction Type consists of three parts:

- Central Module
- Server Type
- Transaction Number

Central Module

The Central Module defines which subsystem handles the transaction. The following central modules exist:

- M = Matching Engine
- C = Deal Capture
- I = Information
- D = Common Database
- U = Supervision

Server Type

The Server Types describes the type of the operation that the transaction will generate. The following server types exist:

- O = Order
- Q = Query
- A = Answer
- D = Deal
- C = Command
- I = Information
- R = Query

Transaction Number

The Transaction Number is a numerical value used to distinguish between different Transaction Types.

3.2.1.3 Series

Series is a generic 12-byte financial instrument definition that is used to identify the products traded. Series is a part of most transaction headers, see section *Transaction Headers on page 14*.

Depending on the transaction type, there are different requirements on which subfields of the Series definition that must be filled in. Variants are:

- Instrument Type – consists of:

  - Country Number
  - Market Code
  - Instrument Group
- Instrument Class – consists of:
  - Country Number
  - Market Code
  - Instrument Group
  - Commodity Code
- Complete Series – all 12 bytes

If not all subfields are filled in, the rest should be filled with binary zeroes.

3.2.1.4 Series Definition

Series are defined by the Exchange and can be divided into five important entities:

Entity	Description
Instrument Type	3 bytes Consists of: • Country Number • Market Code • Instrument Group
Modifier	1 byte
Security Code [Commodity Code]	2 bytes
Expiration date	2 bytes Consists of: • Year • Month • Day
Strike Price	4 bytes

The following picture illustrates the Series definition:

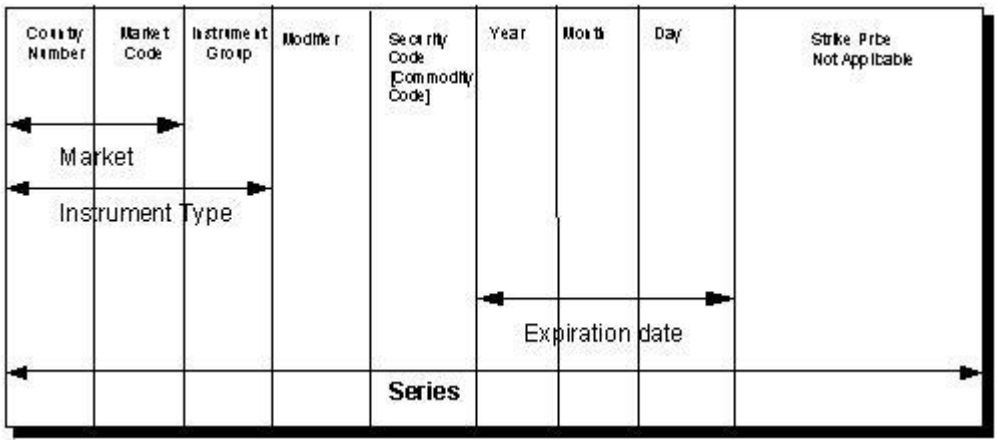

Figure 4: Series Definition

Example 2:

- A Swedish OMX European call option with Strike Price 760 SEK and Expiration Month July 1993 to be routed to OMX Stockholm, is registered as follows:

  0101 0100 0001 08F7 0001 28E0 (hex)

- A US T-Bill can be represented as following:

  1903 2207 D100 0000 0000 0000 (hex)

3.2.1.5 Series Wildcards

For some transactions, typically queries, it is not necessary to state the complete 12 byte Series. Instead, the first three bytes, the Instrument Type, or the first five bytes, the Instrument Class, is often sufficient. The parts of the Series that is not used must be set to 0 (zero).

3.2.1.6 Fillers

In omex.h explicit fillers are used to get a controlled alignment of data structures since different compilers on different platforms might put in implicit (hidden) fillers for unaligned data in data structures. Therefore, a controlled alignment is extremely important when sending binary data streams between different platforms.

3.2.2 Transaction Headers

All transactions are prefaced with a *transaction header*. The header specifies how to interpret the transaction content and how to find the responsible server.

A transaction header can be divided into the following elements:

- Transaction Type (the category to which the transaction belongs)
- Series (the instrument definition)

The format of the transaction headers vary, depending on which facility the transaction uses.

3.3 Partition Handling

To increase the central system capacity, there is a possibility to use several instances of each central server. The transaction load is then divided between the server instances using *partitions.*

Depending on the facility, the data is partitioned in different ways. It is the Exchange's decision when and where partitions are going to be used. Phase 1 is started without dividing the load for the matching process. That means no partition on EP0.

If an API call is marked as partitioned, it says that the receiving process in the subsystem is partitioned. If marked as partitioned=FALSE, there is only one instance of the process. If marked as partitioned=TRUE, there may be several instances of the process.

The subsystems CL, DC, IN and ME may be partitioned. CL and DC are partitioned on Instrument Type. ME and IN are partitioned on Country, Market and Commodity.

In CL and DC, you must handle partitioned API calls manually. Thus, when you do a partitioned API call only the target partition will process the API call. If you need information in another partition, you must issue the API call again, but with a new Instrument Type belonging to another partition.

A transaction will always be sent to one partition. A query will be sent to one partition, and it will then automatically point to the next partition.

3.3.1 Partitioning in Genium INET Clearing

In Genium INET Clearing, each node handles its own set of instruments. An example of how to program with Genium INET partitions is described below.

Example 3:

Assume there are three partitions of Clearing and you want to get all positions in Genium INET

Each Clearing node handles a different set of Instrument Types, see table below:

Partition	Instrument Type
1	83-1-1 up to 83-1-255.
2	83-2-1 to 83-2-255.
3	83-4-17 to 83-255-255.

Attention! To get all positions in the clearing system a CQ3 query must be done to every partition.

CQ3 query to partition 1

Specify Instrument Series = 83-1-1-1-1-1-1 in the first CQ3 query. This means this CQ3 query has Instrument Type 83-1-1.Therefore, this CQ3 query will automatically be routed to partition1. In the answer CA3, you will get the Positions in partition1, and also PARTITION_LOW=83-1-1 and PARTITION_HIGH=83-1-255 which reveals which Instrument Types this partition handles.

CQ3 query to partition 2

Next step is to make a new CQ3 with Instrument Series = 83-2-1-1-1-1-1 which will automatically be routed to partition2. In the CA3 answer, you will get the Positions in partition2, and also PARTITION_LOW=83-2-1 and PARTITION_HIGH=83-2-255 which reveals which Instrument Types this partition handles.

CQ3 query to partition 3

Third step is to make a new CQ3 to partition3. However, you cannot specify Instrument Series = 83-3-1-1-1-1-1-1 because partition3 handles all Instrument Types between 83-4-17 to 83-255-255. You must specify Instrument Series = 83-4-17-1-1-1-1.

To be able to know that partition3 starts at 83-4-17 you can make an DQ3 Instrument Type Query. In the DA3, you will see a list of all available Instrument Types. Right after 83-2-255, the next entry will be 83-4-17.

Note: In CQ3 there is a data struct called “SEARCH_SERIES” which is used to select which series the transaction applies to, within a particular partition. This struct accepts wild cards “0” which returns all positions in one partition.

3.3.2 Partitions on EP0, EP3 and EP5

Server processes that are handling transactions on facility EP0, EP3 and EP5 are partitioned using the series data in the transaction header.

By setting the series in the transaction header, the transaction is implicitly sent to the correct server instance. To make filtering possible, some queries contain an additional series field *Series, search.*

The client application can enter an arbitrary series with suitable wildcards to make the central system filter out only relevant series, thus reducing the size of the answer.

Example 4:

Partitioned on EP0

- Country
- Market
- Commodity

Partitioned on EP3 and EP5

- Country
- Market
- Instrument type

Example 5:

The fields *Transaction Type* and *Series* in a CQ transaction are used to route the query to the correct server. The server then uses the field *Series, search* to select the appropriate answer data. If *Series, search* contains wildcards in all sub-fields, all available data for that server is returned.

3.4 General Query Transaction Handling

3.4.1 General

A query transaction will be returned with an answer. The answer is a combination of the data the user is authorized to see and how the query was specified.

If the user is not authorized to see the information or no data could be found, an empty answer will be returned. There are normally no tests done on the validity of the query data.

3.4.2 Segmented and Coupled Queries

The answer to a query may consist of more data than there is place for in one answer transaction. There are two different approaches to handle that:

Using **segmented** queries	This approach is used when querying **static data**, such as CDB data. Since the data is static it is possible to define the segments as “the first segment contains the first 700 items, the second segment contains the next 700 items etc”. More information in *Segmented Queries on page 17*.
Using **coupled** queries	This approach is used when querying **dynamic data**, such as order book data. Since the data is dynamic it is not possible to define different segments. Instead the query contains data where to start when the answer is created, can be seen as a pointer to an item in the order book. More information in *Coupled Queries on page 17*.

3.4.2.1 Segmented Queries

Segmented queries contain a *segment number* in both the query and the answer in order to tell the requestor if there is more data.

A segment number in the answer that equals the segment number in the query indicates that there is more data and that the requestor has to send another query with the *next Segment Number* in order to retrieve the remaining data.

In the initial query the *segment number* shall be set to 1.

If Segment Number 0 is returned in the answer, there is no more data.

If a *segment number* outside the segmentation limits is used, the result buffer will contain no valid data.

3.4.2.2 Coupled Queries

Coupled series contain data in the answer header, such as *Series*, *order number* and *Bid or Ask*, that shall be used in the next query in order to retrieve the remaining data.

The fields *Order Number* and *Bid or Ask* in the query are normally set to 0 (zero) during the initial query. The field *Series* in the query may be set to zero (=0) for all series, or a specific series may be specified.

If the series in the answer is zero-filled, all requested data have been received.

Example 6:

The requestor wants all his orders for all series in the system.

The initial query:

Series: 0

Order Number: 0

Bid or Ask: 0

The answer contains, among others, the following:

Series (used to get next segment): 20 2 6 155 5685 0 10500

Order Number (used to get next segment): 12345

Bid or Ask (used to get next segment), Next: Bid

The second query:

Series: 20 2 6 155 5685 0 10500

Order Number: 12345

Bid or Ask: Bid

3.5 Segmented Broadcasts Handling

Segmented broadcasts contain a *segment number* in order to tell the receiver whether there is more data to receive or not.

In the first broadcast, segment number is 1. Thereafter the segment number is increased by 1 in every broadcast that is sent out. When segment number is 0, there is no more data.

If the segment number in the first broadcast received is higher than 1, the receiver has missed one or more broadcasts.

The table below describes the segment number field in more detail:

Segment number	Description
0	There is no more data (i.e. the last broadcast).
1	The first broadcast (of several).
>1	Broadcasts following the first one.

3.6 Order Handling when the Remote System Terminates

The Central System will classify the Remote System as not reachable when the OMnet software in the Central System no longer can reach the Remote System.

Note: The exact definition of such a non-reachable Remote System ought to be defined in legal regulations.

3.6.1 Logout Timer

If a Remote System terminates without sending a normal logout, the Central System will initiate a timer. After the time delay as defined by the Exchange, all the Remote System's orders will be inactivated.

Once the logout timer has been initiated, it is possible to prevent the orders from becoming inactive in two ways:

- By sending a new login transaction, either from the same Remote System, if possible, or from another system if being authorized. If this is done before the orders have been inactivated, all orders will be kept in the Order Book as if nothing had happened.
- By contacting the Exchange and ask the staff to stop the orders from becoming inactive.

3.6.2 Inactivated Orders

Note: Below is an example on how the concept of inactivated orders can be handled. The policy on how to handle the orders when a remote system goes down is to be defined by the Exchange in legal regulations.

If the orders are inactive, the trader may log in after the orders have become inactive and request the inactive orders to be sent to his Remote System. He or she can then decide if they should be returned to the Central System, with or without any alterations being carried out.

3.7 Error Handling

Assorted errors are returned as an error code that must be translated into text by the Remote System itself.

3.8 Sequences

The following is an **example** of a normal transaction sequence during the day (your exchange may use other transactions):

1. Login Transaction at the Beginning of the day
2. Market Query (DQ7)
3. Instrument Group Query (DQ8)
4. Delta Underlying Query (DQ120)
5. Instrument Type Query (DQ3)
6. Delta Instrument Query (DQ122)
7. Delta Instrument Series Query (DQ124)
8. Combo Series Query (DQ126)
9. Total Order Query (MQ8)
10. Application Status Transaction (UI1)

If the Remote System crashes, add Query Total Inactive Order Transaction (EP0, MQ9) to the sequence above.

3.9 Trading and Instrument Sessions

A Trading Session is sequence of Trading States, used for scheduling the trading day. An Instrument Session is used to handle exceptions in the trading session, for example when there is a market announcement for a certain underlying or series and the Exchange needs to stop trading in that underlying or series for a while.

In this chapter you can read about the hierarchical structure of the Trading Session (see Section 3.9.1 on page 20) and the priority based Instrument Session. Also, here is the definition of the “Active State”, see Section 3.9.3 on page 22. The Active State is the momentary State that is the function of the Trading Session State and the Instrument Session State at any given moment.

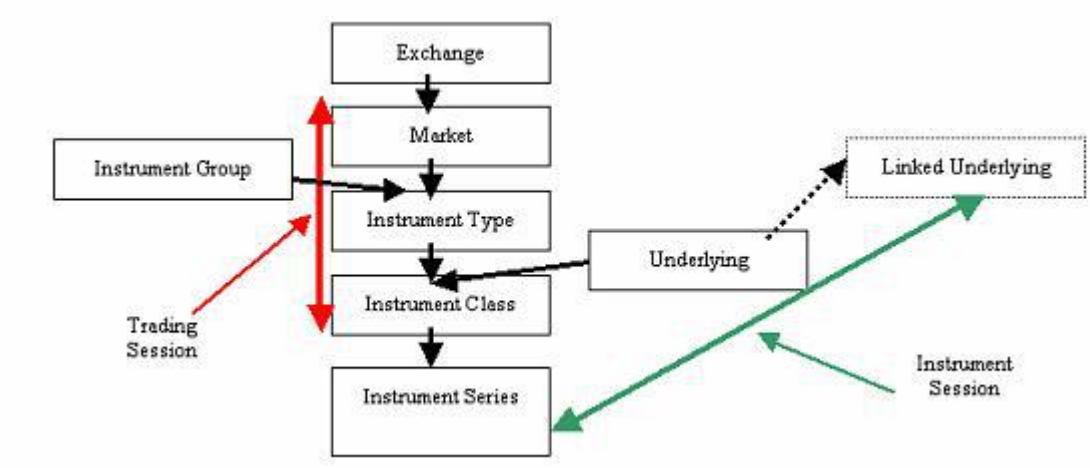

Figure 5: Schematic Overview of Where to Define the Trading Sesson and the Instrument Session

3.9.1 Trading Session

A Trading Session table is a sequence of Trading States, used for scheduling the trading day.

Example 7: Trading Session table

Start Time	Trading State
08:30	Preopen
08:43	Open Allocation
08:45	Open
17:00	Close

Note that the states above and in the examples below are just sample states. Your system configuration may use other states.

A Trading Session table can be connected on three levels:

- Market level

- Instrument Type level
- Instrument Class level

A Trading Session table defined for a lower level overrides the Trading Session defined on a higher level.

The trading state for a given level is either defined explicitly (by an Instrument Status query or an Instrument Status Information broadcast), or implicitly by the first level above it that has an explicit state defined.

Example 8: Trading States Change on Different Levels

This example shows how the trading states change on different levels.

At start-up, an Instrument Status query gives the following answer:

Market/Type/Class	Trading State
Market 1	Open
Instrument Type 11	Preopen

These are the Markets, Instrument Types or Instrument Class that it addresses.

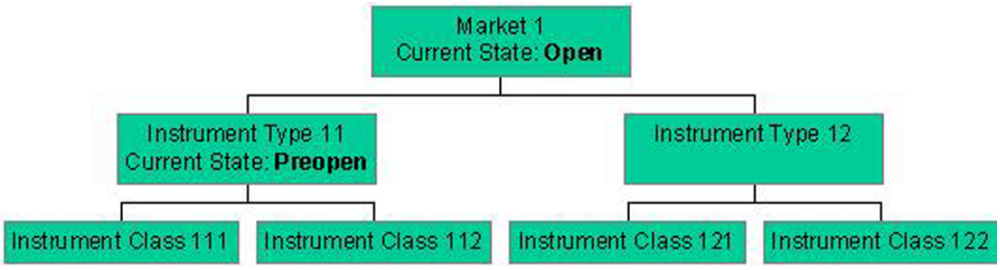

Figure 6: Trading States on Different Levels

When the Instrument Status Information broadcast arrives, it affects only the market, instrument type or instrument class that has a Trading State specified.

The scenario A-D shows what happens when the trading states change.

A: Instrument Status Information broadcast for Market 1 -> **Pause**

The Trading State for Market 1 is changed to Pause. Instrument Type 12, Instrument Class 121 and Instrument Class 122 inherit their trading state from Market 1. That means they change trading state as well.

Instrument Type 11 is not affected by this Trading State change since it is explicitly defined and does not inherit its Trading State.

B: Instrument Status Information broadcast for Instrument Type 11 -> **Pause**

The Trading State for Instrument Type 11 is changed to Pause. Instrument Class 111 and Instrument Class 112 are affected as well.

C: Instrument Status Information broadcast for Instrument Type 12 -> **Pause**

The Trading State for Instrument Type 12 is changed to Pause. Instrument Class 121 and Instrument Class 122 are affected as well.

Note: Once the Trading State of an Instrument Type or an Instrument Class has been changed, this Instrument Type or Instrument Class is not inheriting the Trading State from the level above, even though the Trading State is the same.

D: Instrument Status Information broadcast for Market 1 -> **Close**

The Trading State for Market 1 is changed to Close.

Neither Instrument Type 11 nor Instrument Type 12 is affected by this change since both Instrument Types have their own Trading States specified.

Note: Like this example shows, all levels that have a Trading State defined are saved and must be checked against the “State Change” broadcast.

3.9.2 Instrument Session

Instrument Session is used for exception handling in the trading session.

An instrument session can be set by the exchange on two levels:

- Underlying
- Instrument Series

Note that the Instrument Session also shall be interpreted on the Linked Underlying level, i.e. if Underlying X is configured as Linked Underlying for Underlying Y, the potential Instrument Session of Underlying X shall be taken into consideration for instrument series belonging to Underlying Y, see figure 1.

There is no hierarchy or inheritance between Underlying and Instrument Series. The instrument session state depends on the priority of the instrument session state contributions at the different levels.

3.9.3 Active State

The Active State, at any point in time and for any Instrument Series, is the one of Trading Session State and Instrument Session State that has the highest priority.

It is useful for most applications to know the Active State of every instrument.

To determine the Active State, there is an algorithm to use like this:

1. Determine the Trading Session State and its priority (Instrument Status query and Instrument Status Information broadcast for Trading State and a Trading State query for State Priority.)
2. Determine the Instrument Session State and its priority (Instrument Status query and Instrument Status Information broadcast for Trading State and a Trading State query for State Priority.)
3. Compare the priorities:
4. If the priority of the Instrument Session State is higher than or equal to the priority of the Trading Session State, then the Active State equals the Instrument Session State.
5. If the priority of the Instrument Session State is lower than the priority of the Trading Session State, then the Active State equals the Trading Session State.

This algorithm can be illustrated by pseudo-code like this:

```
TSS: (strictly hierarchical)
Active_TSS := Market_TSS(Series);
If Instrument_Type_TSS(Series) != nil
 Active_TSS := Instrument_Type_TSS(Series);
If Instrument_Class_TSS(Series) != nil
 Active_TSS := Instrument_Class_TSS(Series);
ISS: (priority based)
Active_ISS := Series_ISS(Series);
If Underlying_ISS(Series) > Active_ISS
 Active_ISS := Underlying_ISS(Series);
If Linked_Underlying_ISS(Series) > Active_ISS
 Active_ISS := Linked_Underlying_ISS(Series);
TSS and ISS: (priority based)
If Active_ISS >= Active_TSS
 ActiveState := Active_ISS;
Else
ActiveState := Active_TSS;
```

Example 9:

The following examples have the following Priorities defined for the following Session States:

Session State	Priority
CLOSED	250
HALT	200
LUNCH	150
PRE_OPEN	100
OPEN	50

Note that the states above and in the examples below are just sample states. Your system configuration may use other states.

Example 10:

The Trading Session, based on the Market level, changes from CLOSED to OPEN to CLOSED.

A Company Announcement is received at 11:30; Market Control sets the Underlying in PRE_OPEN, with end time 12:30.

Table 1: Trading Session:

Time	“Start-up”	8:00	11:30	12:30	18:00
Market	CLOSED	OPEN	OPEN	OPEN	CLOSED
Instrument Type					
Instrument Class					
Trading Session States	**CLOSED**	**OPEN**	**OPEN**	**OPEN**	**CLOSED**

Table 2: Instrument Session:

Time	“Start-up”	8:00	11:30	12:30	18:00
Linked Underly-ing					
Underlying			PRE_OPEN		
Series					
Instrument Session State			**PRE_OPEN**		

Table 3: This is the Active Session at any given moment during this day.

Time	“Start-up”	8:00	11:30	12:30	18:00
Active Session State	**CLOSED**	**OPEN**	**PRE_OPEN**	**OPEN**	**CLOSED**

Example 11:

The Trading Session, based on the Market level for an Equity market, moves from CLOSED to OPEN to CLOSED.

The Trading Session, based on the Market level for a Derivative market, moves from CLOSED to OPEN to LUNCH (12:00-13:00) to OPEN to CLOSED.

Note: The diagram below does not show the CLOSED in the beginning and the end of the day.

A Company Announcement is received at 11:30; Market Control sets the Underlying in PRE_OPEN, with end time 12:30.

Table 1: Trading Session for an Equity Market:

Time	8:00	11:30	12:00	12:30	13:00
Market (Equity)	OPEN	OPEN	OPEN	OPEN	OPEN
Instrument Type					
Instrument Class					
Trading Session State	**OPEN**	**OPEN**	**OPEN**	**OPEN**	**OPEN**

Table 2: Trading Session for a Derivative Market:

Time	8:00	11:30	12:00	12:30	13:00
Market (Derivative)	OPEN	OPEN	LUNCH	LUNCH	OPEN
Instrument Type					
Instrument Class					
Trading Session State	**OPEN**	**OPEN**	**LUNCH**	**LUNCH**	**OPEN**

Table 3: Instrument Session:

Time	8:00	11:30	12:00	12:30	13:00
Link Underlying					
Underlying		PRE_OPEN	PRE_OPEN		
Series					
Instrument Session State		**PRE_OPEN**	**PRE_OPEN**		

Table 4: Active States for the Equity and Derivative markets:

Time	8:00	11:30	12:00	12:30	13:00
Active Session State Equity	**OPEN**	**PRE_OPEN**	**PRE_OPEN**	**OPEN**	**OPEN**
Active Session State Derivative	**OPEN**	**PRE_OPEN**	**LUNCH**	**LUNCH**	**OPEN**

3.9.3.1 Active Session State for Combinations

As the different legs of a combo could have different Active States, for example due to that one of its Instruments has triggered a Circuit Breaker, the combinations' Active States must be handled in a special manner.

The Active State of a combination shall be the State with the highest priority among the individual legs of the combo.

If more than one leg of the combo has States with the same highest priority, the combo gets the first of these States, counted in the order the legs of the combo appear in the queries and broadcasts from the central system.

All parts of the system, central system as well as clients shall use the same algorithm to decide this:

```
ActiveStateCombo := ComboLegs(1).ActiveState;
For i := 2 to NumberOfLegs
 If ComboLegs(i).ActiveState.Priority > ActiveStateCombo.Priority
  ActiveStateCombo := ComboLegs(i).ActiveState
```

The Active State of the Combo may change any time the Active State of one of its legs' Active State changes.

3.9.4 Variable Information Messages

Most transactions and broadcasts have fixed sizes and fixed content, specified in the omex.h definition file. This tends to result in a large amount of work for the API users when new data is changed or added to existing transactions or when new transactions are created. To make external API users less dependent on specific API versions, a new way of organizing information is introduced.

The solution is to construct messages with variable size and variable content, depending on which data to distribute. The concept of Variable Information Messages is possible to apply to broadcasts, transactions and queries.

A Variable Information Message contains a variable number of sub-items.

This example (Dedicated Trade Information broadcast BD6) illustrates how a variable information broadcast is built up.

Note: This is an example, BD6 may not be applicable to your exchange.

Code example: Variable Information Broadcast with 2 sub-items	Comments

`struct broadcast_hdr`
`{`
`  broadcast_type_t broadcast_type;`
`  uint16_t items_n;`
`  uint16_t size_n;`
`}`
`struct sub_item_hdr`
`{`
`  uint16_t named_struct_n;`
`  uint16_t size_n;`
`}`
`struct cl_trade_base_api`
`{`
`  ...`
`}`
`struct sub_item_hdr`
`{`
`  uint16_t named_struct_n;`
`  uint16_t size_n;`
`}`
`struct cl_trade_warrants_api`
`{`
`  ...`
`}`
	`BD6`
	`2 sub-items`
	`total size of broadcast`
	`3 = cl_trade_base_api`
	`size of sub-item (subheader+data)`
	`21 = cl_trade_warrants_api`

Code example: Variable Information Broadcast BD6 with 3 sub-items	Comments

`struct broadcast_hdr`
`{`
`  broadcast_type_t broadcast_type;`
`  uint16_t items_n;`
`  uint16_t size_n;`
`}`
`struct sub_item_hdr`
`{`
`  uint16_t named_struct_n;`
`  uint16_t size_n;`
`}`
`struct cl_trade_base_api`
`{`
`  ...`
`}`
`struct sub_item_hdr`
`{`
`  uint16_t named_struct_n;`
`  uint16_t size_n;`
	`BD6`
	`3 sub-items`
	`total size of broadcast`
	`3 = cl_trade_base_api`
	`size of sub-item (subheader+data)`
	`21 = cl_trade_warrants_api`
	`31 = cl_trade_base_part_id`

```
}
struct cl_trade_warrants_api
{
     ...
}
struct sub_item_hdr
{
     uint16_t named_struct_n;
     uint16_t size_n;
}
struct cl_trade_base_part_id
{
     ...
}
```

This example illustrates how a variable information answer is built up, CA11.

Note:

This is an example, CA11 may not be applicable to your exchange.

Code example: Variable Information Answer CA11 with items, 2 trades	Comments

```
struct answer_api_trade_hdr
{
     transaction_type_t
transaction_type;
     series_t series;
     char from_date_s;
int32_t sequence_first_i;
     uint16_t items_n;
     uint16_t size_n;
}
typedef struct item_hdr
{
     uint16_t items_n;
     uint16_t size_n;
}
struct sub_item_hdr
{
     uint16_t named_struct_n;
     uint16_t size_n;
}
struct cl_trade_base_api
{
     ...
}
struct sub_item_hdr
{
     uint16_t named_struct_n;
     uint16_t size_n;
```

```
CA11
2 trades
2 sub-items
size of trade item (item
header+subitems)
3 = cl_trade_base_api
21 = cl_trade_warrants_api
3 sub-items
3 = cl_trade_base_api
21 = cl_trade_warrants_api
31 = cl_trade_base_part_id
```

```
}
struct cl_trade_warrants_api
{
     ...
}
typedef struct item_hdr
{
     uint16_t items_n;
     uint16_t size_n;
}
struct sub_item_hdr
{
     uint16_t named_struct_n;
     uint16_t size_n;
}
struct cl_trade_base_api
{
     ...
}
struct sub_item_hdr
{
     uint16_t named_struct_n;
     uint16_t size_n;
}
struct cl_trade_warrants_api
{
     ...
}
struct sub_item_hdr
{
     uint16_t named_struct_n;
     uint16_t size_n;
}
struct cl_trade_base_part_id
{
     ...
}
```

Variable Information Messages are interpreted by mapping the appropriate structs on each part the received data. Unknown structs can be ignored. The start of the next item in a Variable Information Message is found by skipping the number of bytes given in the size field of the named struct header.

3.10 The Virtual Underlying Concept

Virtual underlying is a grouping concept that makes the dissemination of information and the subscription of such information more efficient.

Virtual underlying allows the exchange to group certain instruments together. For broadcasts supporting the virtual underlying concept, broadcast updates on instruments belonging to the same virtual underlying can be bundled together, thus reducing the bandwidth and the number of subscriptions required for such broadcasts.

The virtual underlying is used when subscribing for broadcasts and when querying for information. The virtual underlying can be seen as an alias underlying for an instrument. There may exist none, one or several

virtual underlyings in the system. An instrument can be tied to at most one virtual underlying, but must not necessarily use this concept. The exchange defines the virtual underlyings to use, and their contents.

The broadcasts and queries that support the virtual underlying concept is stated as

Virtual Underlying Yes in the broadcast and query description in this document.

When API clients download Permanent Information, the use of virtual underlying is returned in the *Query Instrument Class* answer. If a given instrument class in the answer has no virtual underlying specified (a field value of zero), the instrument class does not use the concept. If, however, the virtual underlying field is filled in, this instrument class uses the concept, and:

1. Updates for instruments belonging to this instrument class will be sent using the virtual underlying. A virtual underlying code shall be used when subscribing to such updates. Broadcasts containing updates for multiple instruments may use bundling with other instruments where their instrument class has the same virtual underlying. This is similar to the bundling mechanism based on the real (non-virtual) underlying, which will be in effect for instruments that do not belong to a virtual underlying alias.
2. Certain queries (those supporting virtual underlying) accept a filter series in the query. This filter can be filled in with a virtual instrument class. The filter is used by the system to reply with instruments matching the virtual underlying.
3. The exchange defines broadcast subscription authorization with the use of virtual underlying, in which case an API client will receive updates on all instrument classes within the virtual underlying using one single subscription. Broadcast subscription is further described in *OMnet Application Programmer's Manual*.

3.11 Clearing

It is possible to receive three types of information from the Clearing System via OMnet:

- Real time information
- Batch information
- Signals

3.11.1 Real time Information

The real time information is received as an event (see *OMnet Application Programmer's Interface Manual*).

3.11.2 Batch Information

The batch information is an answer to an ordinary query. Depending on the amount of data available, more than one query may be needed. Each batch data reply will have a segment number in order to indicate that more data follows. Any user connected to a customer can perform the query and the answer incorporates the entire customer.

If the segment number is equal to zero in the answer, there is no more information. If it is not equal to zero, the query must be repeated.

Even if the segment number is non zero, the following segment may be empty. This occurs when the last record in a segment also is the last record available.

Example 12:

- Only one buffer available:

  Query segment number = 1 Answer segment number = 0
- Three buffers available:

  Query segment number = 1 Answer segment number = 1

  Query segment number = 2 Answer segment number = 2

  Query segment number = 3 Answer segment number = 0

3.11.3 Signals

Signals are common events (general broadcasts) indicating that information is available. If more than one type of information is available at the same time, it will be indicated in the same signal.

3.12 Deal and Trade Numbers

For each deal the Central System will create a unique deal number within the instrument type.

Each participating trade in a deal will be given a trade number.

Example 13:

Genium INET has matched the following orders:

- Customer A sells 10 index options for SEK 10
- Customer B buys 5 index options for SEK 10
- Customer C buys 5 index options for SEK 10

When this deal enters the Clearing System, all three belong to the same deal number, but each part in the deal will also have a unique trade number within the instrument type.

In the example above there will be one deal number and three trade numbers.

Example 14:

Genium INET has matched the following orders:

- Customer A sells 10 index options for best possible price
- Customer B buys 5 index options for SEK 10
- Customer C buys 5 index options for SEK 9

This will result in two deals in the Clearing System and therefore two deal numbers and four trade numbers will be created.

- Deal 1: Customer A sells 5 options for SEK 10
- Customer B buys 5 options for SEK 10
- Deal 2: Customer A sells 5 options for SEK 9

Customer C buys 5 options for SEK 9

3.12.1 Identification of Exercises

When an exercise takes place (order or automatically) there will be one part that exercises and one or more parts that are assigned. The assigned parts are drawn by a random algorithm.

A deal number identifies the exercise, i.e. all parts in an exercise get the same deal number.

A trade number uniquely identifies one part of an exercise within an instrument type.

Example 15:
- Customer A orders an exercise of 10 options.
- Customer B and C are randomly selected to deliver underlyings for 5 options each as a result of A's exercise.

This results in one deal number and three trade numbers.
- Deal number = 2957
- Customer A's trade number = 10521
- Customer B's trade number = 10522
- Customer C's trade number = 10523

When describing the actual delivery between two parts it can look like this:
- Customer A trade number = 10521 receives 5 x number of underlyings/options from X
- Customer A trade number = 10521 receives 5 x number of underlyings/options from X
- Customer B trade number = 10522 delivers 5 x number of underlyings/options to X

Customer C trade number = 10523 delivers 5 x number of underlyings/options to X

3.13 Order Transactions

The order-related transactions are using an OMnet facility called EP0, see *OMnet Application Programmer's Interface Manual* for details.

Order related transactions could be divided into three groups:

1. The placing of different types of orders.
2. Alter an order.
3. Deleting an order or a group of orders.

Group	Transaction Type	Is Used…
1	Order Entry	When an order is placed either as a single order or as a Standard Combination Order.
	Trade Report	In a Trade Report.
	Two-Sided Price Quotation	In a Market Maker Order.
2	Alteration	When altering an order in the Order Book.
3	Order Deletion	When deleting an order in the Order Book.

3.13.1 Order Number

When an order transaction is sent, an order number is returned to the sender. This order number is used when information is circulated, but it is encoded so that others cannot identify the sender.

The order number together with the series and the bid/ask flag uniquely identifies an order/quote.

See also the OMnet Order Identification (ordidt) in *OMnet Application Programmer's Interface Manual.*

3.13.2 Order Transactions and Queries with Trader ID

Trader ID transactions (also known as proxy or on-behalf transactions) are used when a trader, user or application wants to send a transaction on behalf of someone else. For example there can be a server that takes care of all the transactions sent to the exchange. In this case, the Trader ID transactions can be used to trace the original user behind the order. These transactions have the same transaction name as the "normal" transactions, but 384 is added to the number. Thus, a transaction called MO31 transaction would have a corresponding MO415 Trader ID transaction.

The only thing that differentiates these trader ID transactions from their respective base transaction is an extra field called trading_code. This field must be filled with the trading code of the user the Trader ID transaction or query is sent for.

All the answers from the transactions/queries are the same as for the base transactions.

For the queries, user_code_t is used instead of trading_code_t in the structs. Note that some queries are used for querying orders entered for another user. An example of this is the Proxy Order query, and consequently, this query does not have a base query, since these are used for these new (+384) transactions only.