Moscow Exchange

Market Data Multicast
FIX/FAST Platform

User Guide

Moscow Exchange
Version 3.3.3
September 29, 2014

Contents

1. Overview

This document describes the Moscow Exchange (identified as MOEX below) MOEX Market Data Multicast FIX/FAST Platform. This platform provides the new highly efficient mechanism for delivering MOEX Market Data to market data consumers. The mechanism utilizes the FIX protocol for messages structure and syntax, FAST protocol for optimization of data streaming, and UDP protocol for delivering data to multiple users efficiently.

MOEX Market Data Multicast FIX/FAST Platform includes the following aspects: streaming data, incremental messaging, FIX format, FAST compression, multicast delivery, and recovery.

1.1. Document History

Issue	**Date**	**Description**
1.0	May 25, 2011	Original version of this document
2.0	December 12, 2012	Clarifications added
3.3	April 08, 2013	Negotiated and REPO deals – specific fields added Message format changes to separate SECBOARD, Trading Status, and Trading Period in individual tags. Additional fields to support REPO with CCP, Closing Auctions, Discrete Auctions, Dark pool auctions, T+2 trading data New FAST compression template Improved readability and fixing document's errata
3.3.1	May 24, 2013	Fixing document errors and adding clarifications per users' feedback. Removing unused fields from document. Compression template has been corrected. Document has revision marks ON to highlight changes.
3.3.2	September 04, 2013	Updated specifications for units (lots or securities) that are used in trading volumes (271)
3.3.3	March 26, 2014	Added field, due to changes in the Listing Rules.
	September 29, 2014	Fixed inaccuracies, removed unused fields and values.

1.2.Streaming Data

Streaming data is the model which allows one to compose a continuous sequence of information of determinate length into one message. It is promote to decrease latency and provide very high volume data routing.

1.3.Incremental Messaging

Incremental data model clearly provides less wasteful on resources. Minimum numbers of instructions are needed to update the book: add, change, delete. An incremental approach sends only necessary data of market events and is intended to significantly reduce data content.

1.4.FIX Format

MOEX Market Data Multicast FIX/FAST Platform uses FIX message format for messages structure and syntax. Message fields are delimited using the ASCII 01 <SOH> character. They are composed of a header, a body, and a trailer.

For more information about used messages and tags, see section 4. FIX Message Specification .

1.5.FAST Compression

FAST is a binary compression algorithm used to purpose of the optimization of FIX messages. FAST benefits include reduced bandwidth and reduced latency. They are achieved at the expense of increased processing time and more complex processing algorithms.

The FAST Protocol uses the following approaches to compact data messages:

- implicit tagging;
- field encoding;
- presence map;
- stop bit;
- binary encoding.

These approaches assume that the structures of the transferred messages as well as encoding rules are agreed between the counter parties. This is usually done via the exchange of machine readable XML-based FAST templates.

For more information about FAST Implementation in MOEX Market Data Multicast, see section 3.2. FAST Implementation.

1.6.Multicast Delivery

Messages are disseminated over the UDP protocol, which allows the Platform to transfer a single packet to multiple destinations and provides lower than TCP transmission latency.

One FAST encoded FIX message does not occupy more than one UDP packet. This ensures the feed is optimized for bandwidth efficiency by reducing the impact of multiple network headers and provides support for FAST field encoding to utilize the full suite of operators including Increment and Copy. These operators will only be used across a set of messages within a single packet.

Currently MOEX Market Data Multicast FIX/FAST Platform does not send more than one FAST encoded FIX message in a UDP packet, but such possibility can be added in future releases.

To minimize confusion MOEX Market Data Multicast FIX/FAST Platform sends messages from different tables of the Trading System to different multicast groups.

1.7.Recovery

Rapid recovery is increasingly important as clients must be always in the market. Recovery processes are very useful for recipients to minimize the probability of a data loss.

MOEX Market Data Multicast FIX/FAST Platform provides data recovery in two ways:

- Market data recovery using market snapshots – suitable for the recovery of a large-scale data loss (i.e. late joiner or major outage);
- TCP Replay of the sent messages – suitable for the recovery of a small-scale data loss (in case when some messages are lost during the transfer).

2. Getting Started with MOEX Market Data FIX/FAST Multicast Platform

2.1. Basic Scenario – Connect before the Trade Day Started

In general, clients should start listening to MOEX Market Data Multicast FIX/FAST Platform some time before the trading day starts. This ensures that client will start receiving actual market data without performing any recovery process.

The procedure is the following:

1. Download the actual multicast IP addresses configuration file from ftp. Configuration file is the XML-file describing the connectivity parameters (feeds multicast addresses, ports, etc.).
2. Download the FAST template from ftp. See section 3.2.5 for the description of the FAST template.
3. Start listening Incremental Feed(s) and sequentially apply received data.

2.2. Connect after the Trade Day Started

If client starts listening to MOEX Market Data Multicast FIX/FAST Platform sometime after the trading day started, it should keep the following procedure:

1. Download the actual multicast IP addresses configuration file from ftp. Configuration file is the XML-file describing the connectivity parameters (feeds multicast addresses, ports, etc.).
2. Download the FAST template from ftp. See section 3.2.5 for the description of the FAST template.
3. Start listening Incremental Feed(s) and queue received data.
4. Start listening Recovery Feed(s), receive and apply actual market data snapshot.
5. Stop listening Recovery Feed(s).
6. Apply queued incremental data.
7. Continue receiving and normal processing incremental data.

2.3. Incremental Feeds A and B Arbitration

Data in all UDP Feeds are disseminated in two identical feeds (A and B) on two different multicast IPs. It is strongly recommended that client receive and process both feeds because of possible UDP packet loss. Processing two identical feeds allows one to statistically decrease the probability of packet loss.

It is not specified in what particular feed (A or B) the message appears for the first time. To arbitrate these feeds one should use the message sequence number found in Preamble or in tag 34-MsgSeqNum. Utilization of the Preamble allows one to determine message sequence number without decoding of FAST message.

Processing messages from feeds A and B should be performed using the following algorithm:

1. Listen feeds A and B
2. Process messages according to their sequence numbers.
3. Ignore a message if one with the same sequence number was already processed before.
4. If the gap in sequence number appears, this indicates packet loss in both feeds (A and B). Client should initiate one of the Recovery process. But first of all client should wait a reasonable time, perhaps the lost packet will come a bit later due to packet reordering. UDP protocol can't guarantee the delivery of packets in a sequence.

Example:

Feed A
34-MsgSeqNum = 59
34-MsgSeqNum = 60
34-MsgSeqNum = 62
34-MsgSeqNum = 63
34-MsgSeqNum = 65

Feed B
34-MsgSeqNum = 59
34-MsgSeqNum = 60
34-MsgSeqNum = 61
34-MsgSeqNum = 62
34-MsgSeqNum = 65

Messages are received from Feed A and Feed B.

1. Receive message # 59 from Feed A, process it.
2. Receive message #59 from Feed B, discard it, because this message was processed before from Feed A.
3. Receive message # 60 from Feed A, process it.
4. Receive message #60 from Feed B, discard it, because this message was processed before from Feed A.
5. Receive message #62 from Feed A, discard it and wait for message #61.
6. Receive message # 61 from Feed B, process it.
7. Receive message # 62 from Feed B, process it.
8. Receive message #62 from Feed A, discard it, because this message was processed before from Feed B.
9. Receive message # 63 from Feed A, process it.
10. Receive message #65 from Feed A, discard it and wait for message #64.
11. Receive message #65 from Feed B, discard it and wait for message #64.
12. Begin recovery process, because gap is detected. Message #64 is missed.

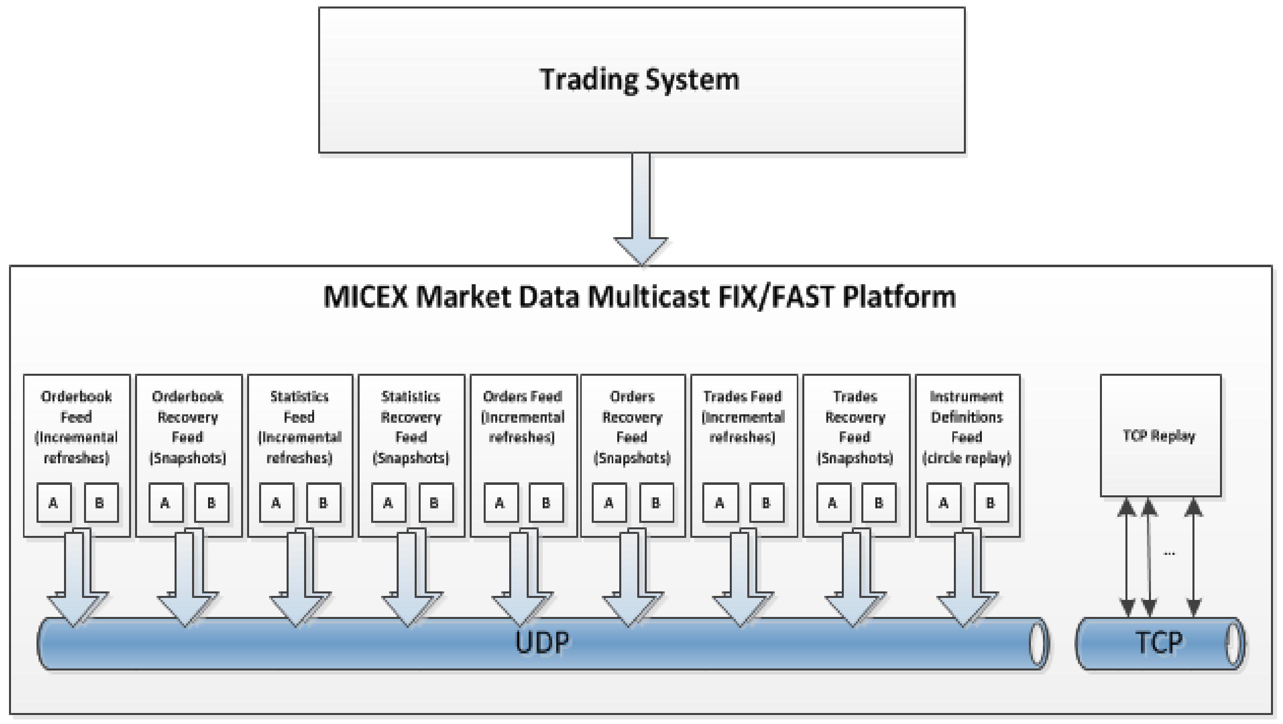
3. Core Functionality

3.1.Platform Architecture

UDP channels used to transfer market data from MOEX. UDP channels are also used for recovery process, TCP connection is used to replay sets of lost messages, already published in the one of UDP Channels.

Following feeds are used in the system:

1. Basic:
   1.1. Market Data Incremental Refresh feeds.
   1.2. Instrument Definition feed.
2. Recovery:
   2.1. Market Recovery feed.
   2.2. TCP Replay session.


Trading System
MICEX Market Data Multicast FIX/FAST Platform
Orderbook Feed (Incremental refreshes)
Orderbook Recovery Feed (Snapshots)
Statistics Feed (Incremental refreshes)
Statistics Recovery Feed (Snapshots)
Orders Feed (Incremental refreshes)
Orders Recovery Feed (Snapshots)
Trades Feed (Incremental refreshes)
Trades Recovery Feed (Snapshots)
Instrument Definitions Feed (circle replay)
A
B
TCP Replay
...
UDP
TCP

MOEX Market Data Multicast broadcast feeds:

- Basic Feeds:
  - Aggregated OrderBook Feeds (OBR), 20 best price levels for buy and for sell
    - OrderBook Feed A
    - OrderBook Feed B
  - Market Statistics Feeds (MSR)
    - Statistics Feed A
    - Statistics Feed B
  - Active Orders List Feeds (OLR)
    - Orders Feed A
    - Orders Feed B
  - Trades List Feeds (TLR)
    - Trades Feed A

        - Trades Feed B
- Recovery Feeds:
    - Aggregated OrderBook Recovery Snapshot Feeds (OBS)
        - OrderBook Recovery Feed A
        - OrderBook Recovery Feed B
    - Market Statistics Recovery Snapshots Feeds (MSS)
        - Statistics Recovery Feed A
        - Statistics Recovery Feed B
    - Active Orders List Recovery Snapshots Feeds (OLS)
        - Orders Recovery Feed A
        - Orders Recovery Feed B
    - Trades List Recovery Snapshot Feeds (TLS)
        - Trades Recovery Feed A
        - Trades Recovery Feed B
- Instruments Definitions Feeds (IDF):
    - Instruments Definitions Feed A
    - Instruments Definitions Feed B

Besides publishing market data in UDP channels, MOEX Market Data Multicast FIX/FAST Platform can accept TCP requests from clients. The replay of data from the following feeds can be requested over TCP connection:

- OrderBook Feed (OBR)
- Statistics Feed (MSR)
- Orders Feed (OLR)
- Trades Feed (TLR)

There are some restrictions for market data transfer over TCP connection:

1. market data messages may be available for limited original publishing time interval back from the time of request;
2. the number of messages, which can be requested over the one TCP session, is limited by 500 messages per request;
3. the number of messages, which can be requested through TCP-replay during the trading day, may be limited.

3.2.FAST Implementation

This part contains the description of the implementation FIX Adapted for STreaming (FAST) protocol.

3.2.1 Introduction

The FIX Adapted for STreaming (FAST) Protocol has been developed as part of the FIX Market Data Optimization Working Group. FAST is designed to optimize electronic exchange of financial data, particularly for high volume, low latency data dissemination.

FAST is a data compression algorithm that significantly reduces bandwidth requirements and latency between sender and receiver. FAST works especially well at improving performance during periods of peak message rates. FAST extends the base FIX specification and assumes the use of FIX message formats and data structures. FAST is a standalone specification that uses templates to encode an instance of an application type, or part thereof, as a stream of bytes, and to inform the receiver which operations to use in decoding.

MOEX Market Data Multicast Platform distributes FIX messages which are encoded in FAST. The Preamble is found before the FAST encoded message, and contains the sequence number (Fig 1).

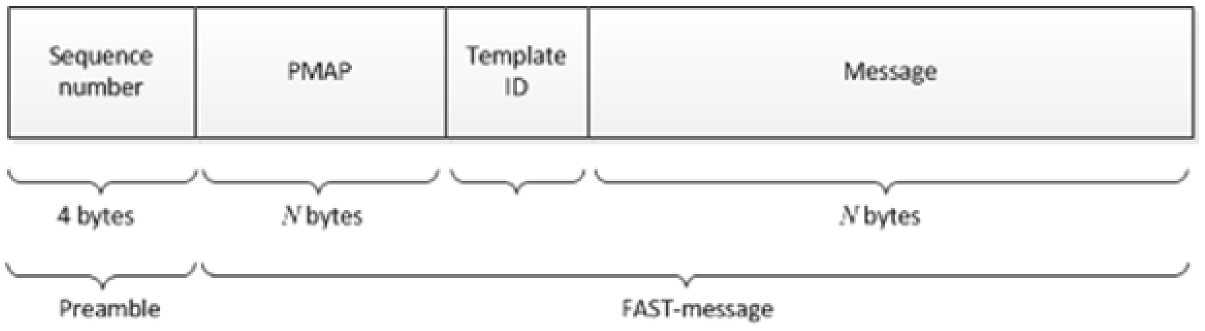


Figure 1

3.2.2 Stop Bit Encoding

An important property of the FAST transfer encoding is the use of stop bit encoded entities. In FAST, a stop bit is used instead of FIX's traditional <SOH> separator byte. Thus 7 bits of each byte are used to transmit data and the eighth bit is used to indicate the end of a field.

3.2.3 Implicit Tagging

In traditional FIX messages each field takes the form "Tag=Value<SOH>" where the tag is a number representing which field is being transmitted and the value is the actual data content. The ASCII <SOH> character is used as a byte delimiter to terminate the field. For example:

35=x|268=3 (message header)
279=0|269=2|270=9462.50|271=5|48=800123|22=8 (trade)
279=0|269=0|270=9462.00|271=175|1023=1|48=800123|22=8|346=15 (new bid 1)
279=0|269=0|270=9461.50|271=133|1023=2|48=800123|22=8|346=12 (new bid 2)

FAST eliminates redundancy with a template that describes the message structure. This technique is known as implicit tagging as the FIX tags become implicit in the data. A FAST template replaces the tag=value syntax with "implicit tagging" as follows:

• tag numbers are not present in the message but specified in the template
• fields in a message occur in the same sequence as tags in the template
• the template specifies an ordered set of fields with operators.

3.2.4 Field Encoding Operators

FAST functions as a state machine and must know which field values to keep in memory. FAST compares the current value of a field to the prior value of that field and determines if the new value should be constant, default, copy, delta (integer or string), increment, or tail.

Some operators rely on a previous value. A dictionary is a cache in which previous values are maintained. All dictionary entries are reset to the initial values specified after each UDP packet. Currently, MOEX sends one message per UDP packet. In this realization delta is not needed.

A field within a FAST template will generally have one of the Field Operators: Constant, Default, Copy, Delta, Increment.

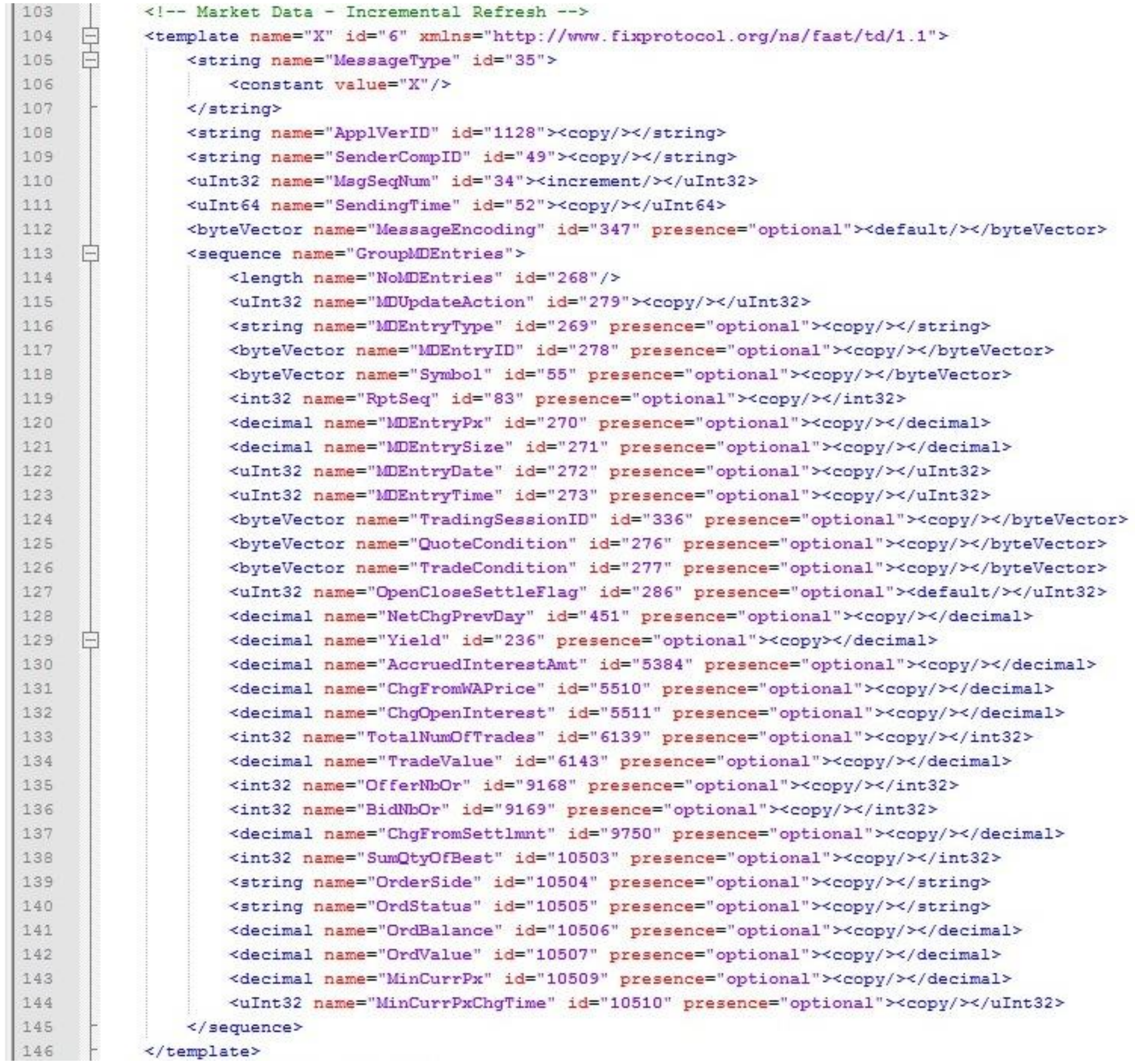
A field within a FAST template will have one of the following Data Types: String, Signed Integer, Unsigned Integer, byte Vector, and Decimal.

3.2.5 FAST Template

A FAST template corresponds to a FIX message type and uniquely identifies an ordered collection of fields. The template also includes syntax indicating the type of field and transfer decoding to apply. A template is communicated between MOEX and client systems in XML syntax using the FAST v1.1 Template Definition Schema maintained by FIX. The XML format is human- and machine-readable and can be used for authoring and storing FAST templates. Session Control Protocol (SCP) will not be used.

A template consists of Field Instructions that define the fields contained in the message. Field Instructions specify the field name, tag number, data type, field operator, and presence attribute that indicate if a field is optional or mandatory.

A sample market data template is shown below (Fig. 2). The syntax is standard XML and can be parsed using a variety of open source tools. Valid template syntax is determined by the FAST Template Schema which is available in the FAST v1.1 specification.



```xml
<!-- Market Data - Incremental Refresh -->
<template name="X" id="6" xmlns="http://www.fixprotocol.org/ns/fast/td/1.1">
    <string name="MessageType" id="35">
        <constant value="X"/>
    </string>
    <string name="ApplVerID" id="1128"><copy/></string>
    <string name="SenderCompID" id="49"><copy/></string>
    <uInt32 name="MsgSeqNum" id="34"><increment/></uInt32>
    <uInt64 name="SendingTime" id="52"><copy/></uInt64>
    <byteVector name="MessageEncoding" id="347" presence="optional"><default/></byteVector>
    <sequence name="GroupMDEntries">
        <length name="NoMDEntries" id="268"/>
        <uInt32 name="MDUpdateAction" id="279"><copy/></uInt32>
        <string name="MDEntryType" id="269" presence="optional"><copy/></string>
        <byteVector name="MDEntryID" id="278" presence="optional"><copy/></byteVector>
        <byteVector name="Symbol" id="55" presence="optional"><copy/></byteVector>
        <int32 name="RptSeq" id="83" presence="optional"><copy/></int32>
        <decimal name="MDEntryPx" id="270" presence="optional"><copy/></decimal>
        <decimal name="MDEntrySize" id="271" presence="optional"><copy/></decimal>
        <uInt32 name="MDEntryDate" id="272" presence="optional"><copy/></uInt32>
        <uInt32 name="MDEntryTime" id="273" presence="optional"><copy/></uInt32>
        <byteVector name="TradingSessionID" id="336" presence="optional"><copy/></byteVector>
        <byteVector name="QuoteCondition" id="276" presence="optional"><copy/></byteVector>
        <byteVector name="TradeCondition" id="277" presence="optional"><copy/></byteVector>
        <uInt32 name="OpenCloseSettleFlag" id="286" presence="optional"><default/></uInt32>
        <decimal name="NetChgPrevDay" id="451" presence="optional"><copy/></decimal>
        <decimal name="Yield" id="236" presence="optional"><copy></decimal>
        <decimal name="AccruedInterestAmt" id="5384" presence="optional"><copy/></decimal>
        <decimal name="ChgFromWAPrice" id="5510" presence="optional"><copy/></decimal>
        <decimal name="ChgOpenInterest" id="5511" presence="optional"><copy/></decimal>
        <int32 name="TotalNumOfTrades" id="6139" presence="optional"><copy/></int32>
        <decimal name="TradeValue" id="6143" presence="optional"><copy/></decimal>
        <int32 name="OfferNbOr" id="9168" presence="optional"><copy/></int32>
        <int32 name="BidNbOr" id="9169" presence="optional"><copy/></int32>
        <decimal name="ChgFromSettlmnt" id="9750" presence="optional"><copy/></decimal>
        <int32 name="SumQtyOfBest" id="10503" presence="optional"><copy/></int32>
        <string name="OrderSide" id="10504" presence="optional"><copy/></string>
        <string name="OrdStatus" id="10505" presence="optional"><copy/></string>
        <decimal name="OrdBalance" id="10506" presence="optional"><copy/></decimal>
        <decimal name="OrdValue" id="10507" presence="optional"><copy/></decimal>
        <decimal name="MinCurrPx" id="10509" presence="optional"><copy/></decimal>
        <uInt32 name="MinCurrPxChgTime" id="10510" presence="optional"><copy/></uInt32>
    </sequence>
</template>
```

Figure 2

3.2.6 Decoding overview

The FAST template contains the instructions to decode and reconstruct compressed message data into the FIX format and also supports repeating groups (sequences) that allow a single message to convey multiple instructions (i.e. book update, trade, high/low, etc.).

Decoding process include following steps:

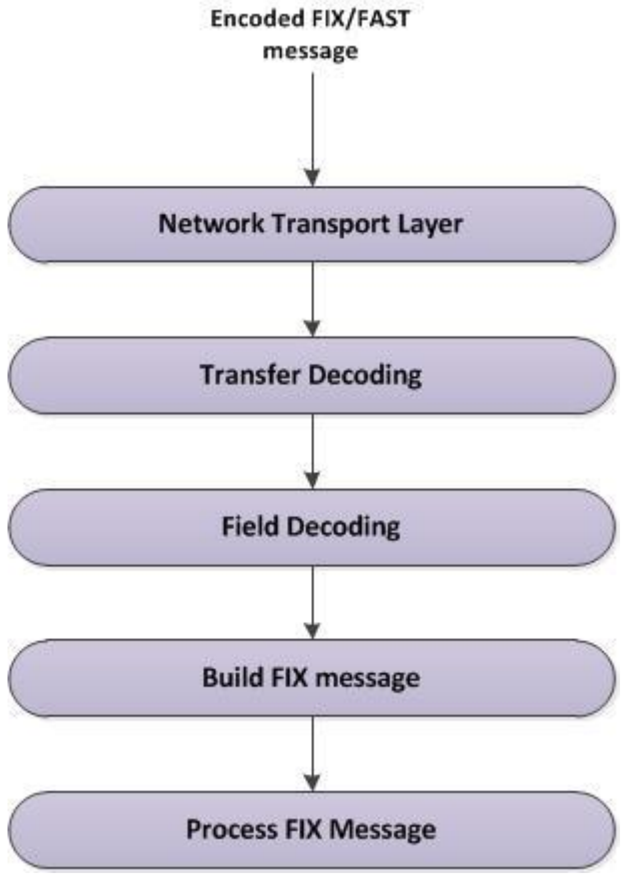


Figure 3

- Transport.

  Client System receives encoded FAST message.

- Transfer decoding.

  Transfer decoding is the initial step that converts data from the FAST 7-bit binary format. It includes:

  - Identify template;

    - Extract binary encoded bits;
    - Map bits to fields per template.
- Field decoding.
  Field decoding is the second part of the decompression process that reconstructs data values according to template-specified operations. Field decoding operations are assigned per field within the template; decoding reinstates data as indicated by the template.
- Build FIX message.
  It includes:
    - Decoding begins with the identification of the Pmap bit for each field.
    - The encoded FAST 7-bit binary values are obtained.
    - Then the encoded FAST 7-bit binary values are de-serialized based on the data type specified in the template.
    - The decoder maintains the state of prior values for each field throughout decoding and applies them for fields having operators of Delta, Copy, or Increment.
    - Obtain fully decoded values.
- Process FIX message.

3.2.7 Sample Template

Table 1

LLine #	**Template Syntax**	**Use and Description**
1	<template name="X" id="6" xmlns="http://www.fixprotocol.org/ns/fast/td/1.1">	Provides the template name and template identifier.
2	<string name="MessageType" id="35"> <constant value="X" /> </string>	Field instruction for MessageType defined as a string with identifier = 35 corresponding to the FIX tag number. MessageType has a constant field operator with a value of *X* which indicates the FIX message type—in this case Market Data Incremental Refresh.
3	<string name="ApplVerID" id="1128"><copy/></string>	Field instruction for ApplVerID defined as a string with an identifier = 1128 corresponding to the FIX tag number. ApplVerID has a copy field operator.
4	<string name="SenderCompID" id="49"><copy/></string>	Field instruction for SenderCompID defined as a string with identifier = 49 corresponding to the FIX tag number. SenderCompID has a copy field operator.
5	<uInt32 name="MsgSeqNum" id="34"><increment/></uInt32>	Field instruction for MsgSeqNum defined as an unsigned integer with identifier = 34 corresponding to the FIX tag number. MsgSeqNum has an increment field operator.
6	<uInt64 name="SendingTime" id="52"><copy/></uInt64>	Field instruction for SendingTime defined as an unsigned integer and with identifier = 52 corresponding to the FIX tag number. SendingTime has a copy field operator.
7	<byteVector name="MessageEncoding"	Field instruction for MessageEncoding defined as a byte vector and with identifier =

	id="347"presence="optional"><default/></byteVector>	347 corresponding to the FIX tag number. MessageEncoding has a default field operator.
8	<sequence name="GroupMDEntries"> <length name="NoMDEntries" id="268"/>	Sequence instruction demarks the beginning of the MDEntries repeating group. The sequence includes a length field called ‘NoMDEntries’ that specifies the number of repeating groups present in the message.
9	<uInt32 name="MDUpdateAction" id="279" presence="optional"><copy/></uInt32>	Field instruction for MDUpdateAction defined as an unsigned integer and identifier = 279 corresponding to the FIX tag number. MDUpdateAction has a copy field operator.
10	<string name="MDEntryType" id="269" presence="optional"><copy/></string>	Field instruction for MDEntryType which is defined as a string and has an identifier = 269 which corresponds to the FIX tag number. MDEntryType has a copy field operator.
11	<byteVector name="MDEntryID" id="278" presence="optional"><copy/></byteVector>	Field instruction for MDEntryID which is defined as a byte vector and has an identifier = 278 which corresponds to the FIX tag number. MDEntryID has a copy field operator.
12	<byteVector name="Symbol" id="55" presence="optional"><copy/></byteVector>	Field instruction for Symbol which is defined as a byte vector and has an identifier = 55 which corresponds to the FIX tag number. Symbol has a copy field operator.
13	<int32 name="RptSeq" id="83" presence="optional"><copy/></int32>	Field instruction for RptSeq defined as a signed integer with identifier = 83 corresponding to the FIX tag number. RptSeq has a copy field operator.
14	<decimal name="MDEntryPx" id="270" presence="optional"><copy/></decimal>	Field instruction for MDEntryPx defined as a decimal with identifier = 270 corresponding to the FIX tag number. MDEntryPx has a copy field operator.
15	<decimal name="MDEntrySize" id="271" presence="optional"><copy/></decimal>	Field instruction for MDEntrySize defined as a decimal with identifier = 271 corresponding to the FIX tag number. MDEntrySize has a copy field operator.
16	<uInt32 name="MDEntryDate" id="272" presence="optional"><copy/></uInt32>	Field instruction for MDEntryDate defined as an unsigned integer and identifier = 272 corresponding to the FIX tag number. MDEntryDate has a copy field operator.
17	<uInt32 name="MDEntryTime" id="273" presence="optional"><copy/></uInt32>	Field instruction for MDEntryTime defined as an unsigned integer and identifier = 273 corresponding to the FIX tag number. MDEntryTime has a copy field operator.
18	<byteVector name="TradingSessionID" id="336"presence="optional"><copy/></byteVector>	Field instruction for TradingSessionID which is defined as a byte vector and has an identifier = 336 which corresponds to the FIX tag number. TradingSessionID has a copy field operator.
19	<byteVector name="QuoteCondition" id="276" presence="optional"><copy/></byteVector>	Field instruction for QuoteCondition which is defined as a byte vector and has an identifier = 276 which corresponds to the FIX tag number. QuoteCondition has a copy field operator.
20	<byteVector name="TradeCondition" id="277" presence="optional"><copy/></byteVector>	Field instruction for TradeCondition which is defined as a byte vector and has an identifier = 277 which corresponds to the FIX tag number. TradeCondition has a copy field operator.
21	<byteVector name="OpenCloseSettlFlag" id="286"presence="optional"><copy/></byteVector>	Field instruction for OpenCloseSettlFlag which is defined as a byte vector and has an identifier = 286 which corresponds to the FIX tag number. OpenCloseSettlFlag has a copy field operator.
22	decimal name="NetChgPrevDay" id="451" presence="optional"><copy/></decimal>	Field instruction for NetChgPrevDay defined as a decimal with identifier = 451 corresponding to the FIX tag number. NetChgPrevDay has a copy field operator.

23	<decimal name="AccruedInterestAmt" id="5384"presence="optional"><copy/></decimal>	Field instruction for AccruedInterestAmt defined as a decimal with identifier = 5384 corresponding to the FIX custom tag number. AccruedInterestAmt has a copy field operator.
24	<decimal name="ChgFromWAPrice" id="5510" presence="optional"><copy/></decimal>	Field instruction for ChgFromWAPrice defined as a decimal with identifier = 5510 corresponding to the FIX custom tag number. ChgFromWAPrice has a copy field operator.
25	<int32 name="TotalNumOfTrades" id="6139" presence="optional"><copy/></int32>	Field instruction for TotalNumOfTrades defined as a signed integer with identifier = 6139 corresponding to the FIX custom tag number. TotalNumOfTrades has a copy field operator.
26	<decimal name="TradeValue" id="6143" presence="optional"><copy/></decimal>	Field instruction for TradeValue defined as a decimal with identifier = 6143 corresponding to the FIX custom tag number. TradeValue has a copy field operator.
27	<decimal name="Yield" id="236" presence="optional"><copy/></decimal>	Field instruction for Yield defined as a decimal with identifier = 236 corresponding to the FIX tag number. Yield has a copy field operator.
28	<int32 name="OfferNbOr" id="9168" presence="optional"><copy/></int32>	Field instruction for OfferNbOr defined as a signed integer with identifier = 9168 corresponding to the FIX custom tag number. OfferNbOr has a copy field operator.
29	<int32 name="BidNbOr" id="9169" presence="optional"><copy/></int32>	Field instruction for BidNbOr defined as a signed integer with identifier = 9169 corresponding to the FIX custom tag number. BidNbOr has a copy field operator.
30	<string name="OrderSide" id="10504" presence="optional"><copy/></string>	Field instruction for OrderSide defined as a string with an identifier = 10504. OrderSide has a copy field operator.
31	<string name="OrderStatus" id="10505" presence="optional"><copy/></string>	Field instruction for OrderStatus defined as a string with an identifier = 10505. OrderStatus has a copy field operator.
32	<decimal name="MinCurrPx" id="10509" presence="optional"><copy/></decimal>	Field instruction for MinCurrPx defined as a decimal with identifier = 10509. MinCurrPx has a copy field operator.
33	<uInt32 name="MinCurrPxChgTime" id="10510"presence="optional"><copy/></uInt32>	Field instruction for MinCurrPxChgTime defined as an unsigned integer and identifier = 10510. MinCurrPxChgTime has a copy field operator.

3.3.Data Feeds

The use of incremental FIX market data messaging in combination with FAST compression produces highly optimized feeds which are distributed in UDP channels. Each Feed is transferred over separate multicast-address. Feeds have the following structure:

- OrderBook Feeds
  - OrderBook Feed A
  - OrderBook Feed B
- Statistics Feeds

    - Statistics Feed A
    - Statistics Feed B
  - Orders Feeds
    - Orders Feed A
    - Orders Feed B
  - Trades Feeds
    - Trades Feed A
    - Trades Feed B
  - Instruments Feeds
    - Instruments Definitions Feed A
    - Instruments Definitions Feed B

In Feeds A and B the equal market data information is sent. It provides low probability of packets loss, and reduce the need in recovery processes.

3.3.1 Instruments Feed

Instruments Definitions Feed A/B provides the security main parameters in a Security Definition (d) message and changes to the definition and/or identity of the security. In this feeds FIX messages encoded to FAST are sent repeatedly with fixed time interval. One FIX message contains information about one security.

Message example:

8=FIXT.1.1|9=400|35=d|1128=9|34=1551|460=5|423=2|911=1572|49=MOEX|55=VRSBP|48=RU000A0DPG75|22=4|461=EPXXXX|167=PS|107=Voronezh EnergoSbyt.Comp(pref)|15=RUB|120=RUB|5217=2-01-55029-E|5385=FOND|969=0.001|5508=0.4|7595=18716678|350=54|351="Воронеж.энергосб.комп" ОАО ап|5382=20|5383=ВоронЭнСбп|52=20110503-08:29:32.968|870=2|871=27|872=3|871=8|872=0|1310=1|561=1|1309=1|336=SMAL|10=000|

Note: each security symbol (55) may be traded in several trading boards that differ by rules. Tag 336 indicates <Board>. There may be multiple different Board values for each security symbol. Please treat each combination of tags 55 and 336 in Security definition as a separate entity with separate stream of market data updates.

3.3.2 OrderBook, Market Statistics, Orders, and Trades Feeds

The following market data is also distributed in separate feeds:

- OrderBook Feed A/B – changes in aggregated ORDERBOOK table.

  There are three data blocks included in OrderBook feeds:

  1. *Add* - to create/insert a new price at a specified price level (MDUpdateAction(279) =0);
  2. *Change* - change quantity for a price at a specified price level (MDUpdateAction (279) = 1);
  3. *Delete* - remove a price at a specified price level (MDUpdateAction (279) = 2).

All data blocks are issued for a specified entry type MDEntryType (269) = '0' (Bid), '1' (Offer), ‘J’ (Empty book).

- Statistics Feed A/B – market statistics, changes in SECURITIES table.
  Statistics Feeds also include Add, Change, and Delete blocks. Entry types are:
  '0' (Bid);
  '1' (Offer);
  '2' (Last Trade in Market statistics feed);
  '3' (Index Value);
  '4' (Opening Price);
  '5' (Closing Price);
  '7' (Trading Session High Price);
  '8' (Trading Session Low Price);
  '9' (Trading Session VWAP Price);
  ‘A’ (Imbalance)
  'B' (Trade Volume, expressed in number of securities);
  'J' (Empty book);
  'N' (Session high bid);
  'O' (Session low offer);
  'Q' (Auction Clearing Price);
  ‘W’(Closing auction price);
  ‘c’(Closing auction volume);
  ‘f’ (Volume of buy market orders in closing auction);
  ‘g’(Volume of sell market orders in closing auction);
  'i' (Last bid price);
  'j' (Last offer price);
  'h' (Open period price);
  'k' (Close period price);
  'l' (Market price 2); on FX market – FX fixing price as calculated between 11:59 and 12:00 Moscow time.
  'm' (Market price); On FX market – FX fixing price
  'o' (Official open price);
  'p' (Official current price);
  'q' (admitted quote); On FX market: international FX fixing price
  'r' (Official close price);
  'v' (Total bid volume);

'w' (Total offer volume);
's' (Dark pool Auction price)
'x' (Dark Pool Auction volume)
‘y’ (Accrued coupon yield on the settlement date, in rubles per unit of financial instrument)
'u' (Duration);.

- Orders Feed A/B – changes in ORDERS table.
  Orders Feeds also include Add, Change, and Delete blocks. Entry types are: '0' (Bid), '1' (Offer), ‘J’ (Empty book)
- Trades Feed A/B – changes in TRADES table.
  Trades Feeds include only Add block (MDUpdateAction(279) =0) and custom entry type MDEntryType (269) = 'z' (Trade List).

The Market Data Incremental Refresh (MsgType (35) = X) message encoded to FAST is used for market data transfer. These allows one to update applicable parts of information as necessary, as opposed to refreshing all market data each time there is an update.

Trading Session Status (h) message is used to represent connection status with appropriate MOEX market. When status of connection changed this message is sent into UDP channel. When status of a security is changed Security Status (f) message is sent into UDP channel.

3.3.3 Market Recovery Feeds

Each Market Recovery feed (OrderBook, Statistics, Orders, Trades) sends the Market Data Snapshot / Full Refresh (MsgType (35) = W) messages encoded to FAST. One message contains information about one security. Information in Market Data Snapshot / Full Refresh message includes status of the connection with market (TradSesStatus (340) tag) and changes in status of a security (MDSecurityTradingStatus (1682) tag).

Market Recovery feeds should be used for recovery purposes only. Once the client system has retrieved recovery data, it recommended to stop listening to the Market Recovery feeds.

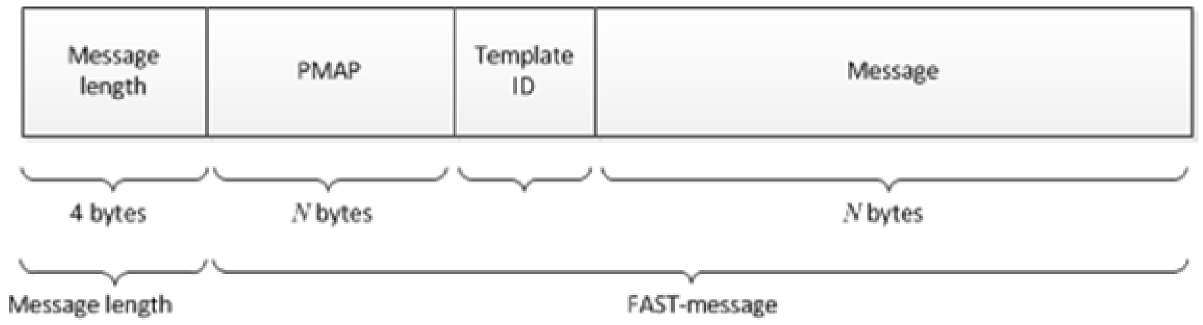
3.3.4 TCP Replay

The TCP replay component allows one to request a replay of a set of messages already published in the one of UDP Channels.

The request is submitted by FIX Market Data Request message (35=V) with range of sequence numbers and UDP Channel identifier.

When establishing TCP-session, client should send the FIX Logon message, always with sequence number 1. When requesting the lost data client should specify the channel ID. Channel IDs can be found in MOEX Market Data Multicast FIX/FAST Platform configuration file available on ftp. They are OLR (for Order List feed), OBR (for OrderBook feed), TLR (for Trade List feed), MSR (for Market Statistics feed).

The length of the message in TCP stream can be found in 4-bytes number before each message being transmitted:



Client can request the limited number of messages. Current limitation is maximum 500 message per request.

Request is sent through a new TCP connection initiated by client. The responses are sent by MOEX Market Data Multicast FIX/FAST through this same connection and the connection is then closed by MOEX Market Data Multicast FIX/FAST once the replay is complete.

TCP Replay should only be used if other options are unavailable. This method has low performance.

3.4.Recovery

MOEX Market Data Multicast FIX/FAST Platform disseminates Market Data in all feeds over two UDP subfeeds: Feed A and Feed B. In Feeds A and B the identical messages are sent. It lowers the probability of packets loss and provides the first level of protection against missed messages.

Sometimes, messages may be missed on both feeds, requiring a recovery process to take place. Message loss can be detected using the FIX message sequence numbers (tag MsgSeqNum (34)), which are also found in the Preamble. The message sequence number is an incrementing number; therefore, if a gap is detected between messages in the tag MsgSeqNum (34) value, or the Preamble sequence number, this indicates a message has been missed. In addition, tag RptSeq (83) can be used to detect a gap between the messages at the instrument level. In this case client system should assume that market data maintained in it is no longer correct and should be synchronized to the latest state using one of the recovery mechanisms.

MOEX Market Data Multicast FIX/FAST Platform offers several options for recovering missed messages and synchronizing client system to the latest state. Market Recovery process together with Instruments Replay Feed is the recommended mechanism for recovery. TCP Replay provides less performance mechanism recommended only for emergency recovering of small amount of lost messages when other mechanisms cannot be used for some reason. Instrument level sequencing and natural refresh can be utilized to supplement the recovery process.

Notes:

- We strongly recommend that client systems process both the A and B Incremental UDP feeds. UDP Feed A and UDP Feed B provide the first level of protection against missed messages.
- We recommend Market Recovery as a primary recovery option.

3.4.1 Market Recovery Overview

This recovery method is preferable to use for large-scale data recovery and for late joiners. Recovery feeds contains Market Data - Snapshot/Full Refresh (W) messages. The sequence number (LastMsgSeqNumProcessed(369)) in the Market Data - Snapshot/Full Refresh (W) message corresponds to the sequence number (MsgSeqNum(34)) of the last Market Data - Incremental Refresh (X) message in the corresponding feed. Instrument level sequence number (RptSeq(83)) in Market Data - Snapshot/Full Refresh (W) message correspond to the sequence number (RptSeq(83)) in the MDEntry from last Market Data - Incremental Refresh (X) message. Thus tag MsgSeqNum(34) shows the gap at the messages level, tag RptSeq(83) shows gap at the instrument level.

After value of RptSeq(83) tag from Market Data - Incremental Refresh (X) becomes more than value of RptSeq(83) tag from Market Data - Incremental Refresh (X), market data becomes actual.

After value of MsgSeqNum(34) from Market Data - Incremental Refresh (X) message becomes more than value of tag LastMsgSeqNumProcessed(369) from Market Data - Snapshot/Full Refresh (W) message, market data becomes actual.

Messages sequence numbers begins from #1 in Market Data - Snapshot/Full Refresh (W) messages in each cycle.

Last Market Data - Snapshot/Full Refresh (W) message in Recovery Feeds sends with tag LastFragment (893) =’Y’.

Clients should keep queuing real-time data until all missed data is recovered. The recovered data should then be applied prior to data queued.

Steps during Recovery process corresponds to the steps 4 – 7 from point 2.2.

Since clients have retrieved recovery data, it is recommended to stop listening Market Recovery feeds.

3.4.2 Recovering Data – Process

The recovering data process should be applied to affected feeds only. Unaffected feeds can be processed as usual. The process can follow two paths: queuing current data while recovering or processing current data while recovering.

3.4.2.1.1. Queuing

This process implies the queuing the Incremental Market Data from Incremental Feeds while receiving Market Data Snapshots from Recovery Feeds. In order to avoid an excessive number of queued messages, it is recommended to process snapshots and apply the applicable incremental feed as the snapshots arrive.

1. Identify Feed(s) in which the client system is out of sync.
2. Listen to and queue the Incremental Market Data from the affected Feed(s).
3. Listen to the Market Recovery Feed corresponding to the affected Incremental Feed(s), receive and apply snapshots.
4. Verify that all snapshots have been received for a given Market Recovery feed, using one of the following approaches:
   a. Message sequence numbers in each loop of snapshots start from 1. So to determine the end of the loop one can wait until the next message with 34-MsgSeqNum = 1 arrives.
   b. Snapshots in the Recovery Feeds are sent in the same order as Security Definitions in Instruments Feed. Tag 893-LastFragment in the W-message indicates if it is the last fragment of the snapshot on the instrument. Receiving the last fragment of the last instrument means the receiving the last snapshot in the loop.

5. Apply all queued incremental data in the sequence, where
    a. tag 34-MsgSeqNum (or the Preamble sequence number) is greater than the lowest value for tag 369-LastMsgSeqNumProcessed;

OR

    b. tag 83-RptSeq from the Market Data Incremental – Refresh message is greater than the lowest value for tag 83-RptSeq on the Market Recovery feed.
6. Continue normal processing

3.4.2.1.2. Concurrent Processing

This process implies the possibility to resume normal processing of an instrument while other affected instruments are still being recovered.

1. Identify Feed(s) in which the client system is out of sync.
2. Listen to the Incremental Market Data from the affected Feed(s) and optionally attempt a natural refresh.
3. Listen to the Market Recovery Feed corresponding to the affected Incremental Feed(s)
4. For each instrument:
    a. compare tag 369-LastMsgSeqNumProcessed on the Market Recovery feed to tag 34-MsgSeqNum (or the Preamble sequence number) on the Incremental Market Data feed and verify that the value for tag 34-MsgSeqNum is not lower;

OR

    b. compare tag 83-RptSeq on the Market Recovery feed to tag 83-RptSeq on the Incremental Market Data feed and verify that the value for tag 83-RptSeq on the Incremental Market Data feed is not lower.
5. Continue normal processing

3.4.2.1.3. Instrument Level Sequencing

Market Data Incremental Refresh messages contain instrument sequence numbers (tag 83-RptSeq), in addition to message sequence numbers (tag 34-MsgSeqNum). Every repeating group instance of a market data entry contains an incrementing sequence number (tag 83-RptSeq) that is associated with the instrument for which the data is present in the block.

Client systems can keep track of the instrument sequence number (tag 83-RptSeq) for every instrument by inspecting incoming data and determining whether there is a gap in the instrument sequence number.

- If there is a gap in the instrument sequence number, it indicates that data was missed for the instrument when message loss occurred.
- If there is no gap, the data can be used immediately, and it also indicates that the book for this instrument still has a correct, current state.

3.4.2.1.4. Natural Refresh

The client system must track the state of the book at all times with the FIX Market Data Incremental Refresh messages. It is possible, though not guaranteed, that a set of these book update messages can be used to construct the current, correct state of a book without prior book state knowledge. This process called Natural Refresh. Prior to beginning a natural refresh, the entire book should be emptied. Natural refresh assumes no prior knowledge of book state. Natural Refresh works best for aggregated orderbook feed and for highly liquid securities.

3.4.3 TCP Replay

If market data from OrderBook, Statistics, Orders, and Trades Feeds was missed, it can be recovered over the TCP historical replay component using the sequence number range. TCP Replay is a low performance recovery option and should only be used if other options are unavailable or for small-scale data recovery. Number of messages which can be requested by client during TCP connection is limited.

TCP replay include follows:

1. Establish TCP connection with MOEX Market Data Multicast.
2. Send FIX message Logon(A) with sequence numder 1 to server. After successful authorization server sends the FAST-encoded Logon(A) message.
3. Send Market Data Request (V) message with:
    a. Tag ApplID (1180) - the channel ID (as specified in server configuration file available on ftp: OLR, OBR, TLR, or MSR).
    b. Range of sequence numbers - ApplBegSeqNum(1182) and ApplEndSeqNum (1183) tags.

If request is correct, server sends FAST messages according to requested sequence numbers.

If request is incorrect, server sends FAST Logout (5) message with reject reason.

After server responses, the connection is closed.

Server will process only first user request, second and others will be ignored. If the server does not receive Market Data Request within an established timeout interval after logon, the connection is closed.

4. FIX Message Specification

This part contains the description of FIX 5.0 SP2 protocol messages, component blocks and fields which are supported by MOEX Market Data Multicast.

This specification is based on FIX 5.0 SP2 standard for application-level messages, FIXT 1.1 for session-level messages (http://fixprotocol.org/) and adapted to MOEX's purposes. It's assumed that users have basic knowledge about FIX standard.

Only messages, component blocks and fields which are described in this document are supported by MOEX Market Data Multicast. Note that all fields which are required or conditionally required by FIX 5.0 SP2 standard but absent in MOEX Interface specification are optional and ***will be ignored by MOEX***. All field values which are valid according to FIX 5.0 SP2 standard but aren't described in this document will be considered as invalid and incoming messages with such values will be rejected.

4.1.FIX Component Blocks

4.1.1 Standard Message Header

Table 2

Tag	Field name	Req'd	Type	Valid values	Comments
1128	AppVerID	Y	String (1)	'9' (FIX50SP2)	Specifies the service pack release being applied for application-level messages.
35	MsgType	Y	String (10)		Defines message type. Always unencrypted.
49	SenderCompID	Y	String (12)		Assigned value used to identify firm sending message. Always unencrypted. If this message is sent to MOEX TCP replay server, SenderCompID may contain arbitrary string.
34	MsgSeqNum	Y	SeqNum		Integer message sequence number.
52	SendingTime	Y	UTCTimestamp		Time of message transmission (expressed in UTC). YYYYMMDD-HH:MM:SS.sss
347	MessageEncoding	N	String(11)	'UTF-8' (Unicode)	Type of message encoding (non-ASCII characters).

					Required if any "Encoding" fields are used.

4.1.2 Instrument

Table 3

Tag	Field name	Req'd	Type	Valid values	Comments
55	Symbol	Y	String(12)		Ticker symbol. The MOEX internal instrument identifier, SecCode. Note: an instrument with a given SecCode may be traded in several trading boards (SecBoard). You should use each Symbol (55)+TradingSessionID (336) combination as an individual security with own order book and list of trades.
48	SecurityID	N	String		Security identifier value of SecurityIDSource (22) type.
22	SecurityIDSource	N	String	'4' (ISIN)	Identifies class or source of the SecurityID (48) value.

460	Product	N	int	'3' (CORPORATE);
				'4' (CURRENCY);
				'5' (EQUITY);
				'6' (GOVERNMENT);
				'7' (INDEX);
				‘10’ (MORTGAGE)
				'11' (MUNICIPAL);
				‘12’ (OTHER);
				'13' (FINANCING).
					Indicates the type of product the security is associated with.

461	CFICode	N	String		Indicates the type of security using ISO 10962 standard, Classification of Financial Instruments (CFI code) values.

167	SecurityType	N	String	*'CORP' (Corporate Bond);*
				'FOR' (Foreign Exchange Contract);
				'CS' (Common Stock);
				'PS' (Preferred Stock);
				'EUSOV' (Euro Sovereigns);'
				MLEG' (Multileg Instrument);
				'MUNI' (Municipal bonds).
				RDR – Russian depositary receipt
				ETF – exchange traded fund
				‘COFP’ (Certificate Of Participation)
					Indicates type of security.

				'XCN' (Extended Comm Note) *'STRUCT' (Structured Notes)* *'WAR' (Warrant)*	
541	MaturityDate	N	LocalMktDate		Maturity date for bonds
224	CouponPayment Date	N	LocalMktDate		Date interest is to be paid.
223	CouponRate	N	Percentage		The rate of interest.
107	SecurityDesc	N	String		Security description.
350	EncodedSecurity DescLen	N	Length		Byte length of encoded (non-ASCII characters) EncodedSecurityDesc (351) field.
351	EncodedSecurity Desc	N	data		Russian language (non-ASCII characters) name for the security. Encoded format is specified via the MessageEncoding (347) field. If used, the ASCII (English) representation should also be specified in the SecurityDesc (107) field.
5217	*StateSecurityID*	*N*	*String*		*State Securities Identification Number.*
5382	*EncodedShortSec urityDescLen*	*N*	*Length*		*Byte length of encoded (non-ASCII characters) EncodedShortSecurityDesc (5383) field.*
5383	*EncodedShortSec urityDesc*	*N*	*data*		*Short (non-ASCII characters) security name in Russian language. Field encoding format specified via the MessageEncoding (347) field.*
5556	*BaseSwapPx*	*N*	*Price*		*Base SWAP price.*
5558	*BuyBackPx*	*H*	*Price*		*Buy back price. Early redemption buyback price for bonds. If defined, the field BuyBackDate must be filled. If defined, yield calculation is based on this date and price.*
5559	*BuyBackDate*	*H*	*LocalMktDate*		*Buy back date. Early redemption of bonds Buyback date. If defined, yield calculation is based on this date.*

4.1.3 Instrument Extension

Table 4

Tag	Field name	Req'd	Type	Valid values	Comments
870	NoInstrAttrib	N	NumInGroup		Number of repeating InstrAttribType (871) entries.
=> 871	InstrAttribType	N	int	'8' (Coupon period); '27' (Instrument Price Precision).	Code to represent the type of instrument attribute. Required if NoInstrAttrib (870) > 0.
=> 872	InstrAttribValue	N	String		Attribute value appropriate to the InstrAttribType (871) field.

4.1.4 Market Segment

Table 5

Tag	Field name	Req'd	Type	Valid values	Comments
1310	NoMarketSegments	N	NumInGroup		Number of Market Segments on which a security may trade.
=> 561	RoundLot	N	Qty		The trading lot size of a security.
=> 1309	NoTradingSessionRules	N	NumInGroup		Allows trading rules to be expressed by trading session.
=> => 336	TradingSessionID	N	String(4)		Identifier for Trading Session. Used to represent SECBOARD. Note: an instrument with a given SecCode may be traded in several trading boards (SecBoard). You should use each Symbol (55)+TradingSessionID (336) combination as an individual security with own order book and list of trades.
=> => 625	TradingSessionSubID	N	String	NA – No trading O – Opening auction period C – Closing period F – Final closing period N – Normal trading period L – Closing auction period I – Discrete auction period D – Dark pool auction period E – Trading at the closing auction price period	Indicates the trading period Notes: • Period is empty before the trading start and after the trading is closed. • Switching between periods typically involves a short stop in trading, in which period is not defined (625=NA) • The sequence and schedule of periods depends on board code and on market conditions as defined by the Exchange Trading rules. • Period value of this component block indicates a period that is running at the start of Security definition publishing cycle. Security status updates that come after Security definitions publishing cycle start should replace tag 625 values from Security definitions feed.
=> => 326	SecurityTradingStatus	N	int	18 – Not available for trading 118 – Opening auction 18 – Trading closed 103 – Closing period	Trading status for a security Notes: • a break in any period is indicated by 326=2 and period

				2 – Break in trading 17 – Normal trading 102 – Closing auction 106 – Dark pool auction 107 – Discrete auction 120 – Trading at Closing auction price	identifier in tag 625. • Not available for trading and Trading Closed are different technological states in the Trading system. However they both disable trading activity and thus have equal values of tag 326. • Trading status value of this component block indicates a trading statsus that existed at the start of Security definition publishing cycle. Security status updates that come after Security definitions publishing cycle start should replace tag 625 values from Security definitions feed.
=>=>9680	OrderNote	N	Char		Level of listing

4.2.FIX Session-Level Messages

4.2.1 Logon (A)

Logon message from customer to MOEX:

Table 6

Tag	Field name	Req'd	Type	Valid values	Comments
<Standard Message Header>		Y			MsgType = 'A'
553	Username	Y*	String		Userid or username.
554	Password	Y*	String		User password.
1137	DefaultApplVerID	Y	String	'9' (FIX50SP2)	Specifies the service pack release being applied, by default, to message at the session level.

Logon message from MOEX to customer:

Table 7

Tag	Field name	Req'd	Type	Valid values	Comments
<Standard Message Header>		Y			MsgType = 'A'

108	HeartBtInt	Y	int		Heartbeat interval (seconds).
1137	DefaultApplVerID	Y	String	‘9’ (FIX50SP2)	Specifies the service pack release being applied, by default, to message at the session level.

4.2.2 Logout (5)

Table 8

Tag	Field name	Req'd	Type	Valid values	Comments
<Standard Message Header>		Y			MsgType = '5'
58	Text	N	String		Logout reason.

4.2.3 Heartbeat (0)

Table 9

Tag	Field name	Req'd	Type	Valid values	Comments
<Standard Message Header>		Y			MsgType = '0'

4.3. FIX Application-Level Messages

4.3.1 Security Definition (d)

Table 10

Tag	Field name	Req'd	Type	Valid values	Comments
<Standard Message Header>		Y			MsgType = 'd'
911	TotNumReports	Y	int		Total number of Security Definition messages in a cycle.
component block <Instrument>		Y			The <Instrument> component block contains all the fields commonly used to describe a security or instrument.
component block <Instrument Extension>		N			The <InstrumentExtension> component block identifies additional security attributes that are more commonly found for Fixed Income securities.

15	Currency	N	Currency		Identifies currency used for price.
component block <Market Segment>		N			Contains all the security details related to listing and trading the security, including its trading status and trading period as they were at the start of Security Definitions publishing cycle. This allows late joiners to get current security trading state if they have missed earlier Security status (35=f) messages.
120	SettlCurrency	N	Currency		Currency code of settlement denomination.
423	PriceType	N	int	'1' (Percentage); '2' (Per unit).	Code to represent the price type. Note: for REPO with CCP this tag value is 1, but indicates the REPO rate, not the price of underlying security (bond or share)
64	*SettlDate*	*N**	*LocalMktDate*		*Specific date of trade settlement (SettlementDate) in YYYYMMDD format* *For Equities and FX in orders driven market: indicates settlement date* *For Equities in quote driven market (negotiated): indicates default settlement date. Actual date may vary and is indicated for each trade in the Trade List feed* *For FX swaps: indicates settlement date for reverse trade.*
5385	MarketCode	N	String		Code of market where instrument is traded. Note: MarketCode indicates a group of trading boards (SECBOARDS) with similar trading rules.
969	MinPriceIncrement	N	float		Minimum price increase for a given exchange-traded Instrument.
5508	FaceValue	N	Amt		Face value of security.
5850	OrigIssuueAmt	N	Int		Number of placed securities in issue
7595	NoSharesIssued	N	Qty		The number of shares issued.

4.3.2 Security Status (f)

Security Status messages indicate changes in current Trading status and period for a security. Please note that publishing multiple 35=f messages in traffic-shaped feeds takes some time, and that this publishing is done in parallel with publishing updates in incremental feeds. Parallel publishing may result in getting an incremental update from new trading state slightly before receiving the status change for a security, or getting an incremental update from previous trading state after the trading status change.

Table 11

Tag	Field name	Req'd	Type	Valid values	Comments

<Standard Message Header>		Y			MsgType = 'f'
83	RptSeq	Y	int		Sequence number of message within report series.
55	Symbol	Y	String		Ticker symbol. The Moscow Exchange internal instrument identifier, SecCode.
336	TradingSessionID	N	String		Identifier for Trading Session. Used to represent SECBOARD. Note: an instrument with a given SecCode may be traded in several trading boards (SecBoard). You should use each Symbol (55)+TradingSessionID (336) combination as an individual security with own order book and list of trades.
625	TradingSessionSubID	N	String	NA – No trading O – Opening auction period C – Closing period F – Final closing period N – Normal trading period L – Closing auction period I – Discrete auction period D – Dark pool auction period E – Trading at the closing auction price period	Indicates the trading period Notes: • Period is empty before the trading start and after the trading is closed. • Switching between periods typically involves a short stop in trading, in which period is not defined (625=NA) • The sequence and schedule of periods depends on board code and on market conditions as defined by the Exchange Trading rules.
326	SecurityTradingStatus	N	int	18 – Not available for trading 118 – Opening auction 18 – Trading closed 103 – Closing period 2 – Break in trading 17 – Normal trading 102 – Closing auction 106 – Dark pool auction 107 – Discrete auction 120 – Trading at Closing auction price	Trading status for a security Notes: • a break in any period is indicated by 326=2 and period identifier in tag 625. • Not available for trading and Trading Closed are different technological states in the Trading system. However they both disable trading activity and thus have equal values of tag 326.
5509	AuctionIndicator	N	Boolean	'Y' (Yes); 'N' (No).	Indicates that the primary distribution auction is being held for the security. Primary distribution auction data is currently not published in the feed. Notes: • 5509=N for ALL other auction types. • Boolean values are encoded in FAST messages as binary

					integers: 1 for Y, and 0 for N.

4.3.3 Trading Session Status (h)

Table 12

Tag	Field name	Req'd	Type	Valid values	Comments
<Standard Message Header>		Y			MsgType = 'h'
336	TradingSessionID	Y	String		Identifier for Trading Session is used to represent SECBOARD.

340	TradSesStatus	Y	int	‘100’ (Connection to MOEX market established);
				‘101’ (Lost connection to MOEX);
				‘102’ (Connection to MOEX market established, trading system wasn't restarted);
				‘103’ (Connection to MOEX market established, trading system was restarted).
					State of the trading session. Informs about connection state between the MOEX Market Data Multicast FIX/FAST Platform and the trading system.
					Note: Receiving the very unlikely message 340=103 means that Trading system has started from scratch and you must remove all feed data on your side and start over.

58	Text	N	String		Free format text string.

4.3.4 Market Data Request (V)

Table 13

Tag	Field name	Req'd	Type	Valid values	Comments
<Standard Message Header>		Y			MsgType = 'V'
1180	ApplID	N	String	OLR, OBR, TLR, MSR	The channel ID.
1182	ApplBegSeqNum	N	SeqNum		Beginning range of application sequence numbers.
1183	ApplEndSeqNum	N	SeqNum		Ending range of application sequence numbers.

4.3.5 Market Data - Snapshot/Full Refresh (W)

Table 14

Tag	Field name	Req'd	Type	Valid values	Comments
<Standard Message Header>		Y			MsgType = 'W'
83	RptSeq	Y	int		Sequence number of message within report series. Value equal to the RptSeq(83) in Market Data - Incremental Refresh (X) message at the time when the snapshot has been prepared.
369	LastMsgSeqNumProcessed	N	SeqNum		Value equal to the MsgSeqNum(34) from the last Market Data - Incremental Refresh (X) message which were received and processed correctly.

340	TradSesStatus	N	int	'100' (Connection to MOEX market established);
				'101' (Lost connection to MOEX);
				'102' (Connection to MOEX market established, trading system wasn't restarted);
				'103' (Connection to MOEX market established, trading system was restarted).
					State of the trading session. Informs about connection state between the MOEX Market Data Multicast FIX/FAST Platform and the trading system.
					Note: Receiving the very unlikely message 340=103 means that Trading system has started from scratch and you must remove all feed data on your side and start over.

55	Symbol	Y	String		Ticker symbol. The MOEX internal instrument identifier, SecCode. Note: an instrument with a given SecCode may be traded in several trading boards (SecBoard). You should use each Symbol (55)+TradingSessionID (336) combination as an individual security with own order book and list of trades.
893	LastFragment	N	Boolean	'N' (Not Last Message); 'Y' (Last Message).	Indicates whether this message is the last in a sequence of messages in the snapshot for a security. Boolean values are encoded in FAST messages as binary integers: 1 for Y, and 0 for N.
1682	MDSecurity Trading Status	N	int	18 – Not available for trading 118 – Opening auction 18 – Trading closed 103 – Closing period 2 – Break in trading 17 – Normal trading 102 – Closing auction	Current trading status for a security Notes: • a break in any period is indicated by 1682=2 • Not available for trading and Trading Closed are different technological states in the Trading system. However they both disable trading

				106 – Dark pool auction 107 – Discrete auction 120 – Trading at Closing auction price	activity and thus have equal values of tag 1682. • Switching between trading periods typically involves a short stop in trading • The sequence and schedule of periods and trading status values depends on SecBoard code (336) and on market conditions as defined by the Exchange Trading rules.
5509	AuctionIndicator	N	Boolean	'Y' (Yes); 'N' (No).	Indicates that the primary distribution auction is being held for the security. Primary distribution auction data is currently not published in the feed. Notes: • 5509=N for ALL other auction types. Boolean values are encoded in FAST messages as binary integers: 1 for Y, and 0 for N.
451	NetChgPrevDay	N	PriceOffset		Net change from previous day’s closing price vs. last traded price.
268	NoMDEntries	Y	NumInGroup		Number of entries in Market Data message.

=> 269	MDEntryType	Y	char	'0' (Bid);
				'1' (Offer);
				'2' (Last Trade in Market statistics feed);
				'3' (Index Value);
				'4' (Opening Price);
				'5' (Closing Price);
				
				'7' (Trading Session High Price);
				'8' (Trading Session Low Price);
				'9' (Trading Session VWAP Price);
				‘A’ (Imbalance), expressed in number of securities
				'B' (Trade Volume, expressed in number of securities);
				
				'J' (Empty book);
				'N' (Session high bid);
				'O' (Session low offer);
				'Q' (Auction Clearing Price), the clearing volume (271) is expressed in lots;
				‘W’(Closing auction price);
					Type Market Data entry.
					Notes:
					• The availability of this field’s values depends on market type (FX or Equities), SecBoard code (336) and the Exchange trading rules.
					• Different feeds have subsets of possible values, depending on the data contents.
					• Empty Book (269=J) indicates no data for a security. Empty Book message may be generated market-wide, which indicates that you should remove all previously collected data and start over.
					• Meaning of some values depend on market type (FX or Equities) and corresponding trading rules
					• Off-book trading boards do not have data in Orderbook Snapshot (OBS) and OrderList snapshot feeds (OLS).
					• Off-book trading boards may have market statistics data for a Symbol taken from on-book

				‘c’(Closing auction volume), expressed in number of securities;
				‘f’ (For MSR/MSS feeds - volume of buy market orders in closing auction, expressed in number of securities; for OLR/OLS – market in closing auction buy order);
				‘g’(For MSR/MSS feeds: volume of sell market orders in closing auction, expressed in number of securities; for OLR/OLS – market in closing auction sell order);
				'i' (Last bid price);
				'j' (Last offer price);
				'h' (Open period price);
				'k' (Close period price);
				'l' (Market price 2); on FX market – FX fixing price as calculated between 11:59 and 12:00 Moscow time.
				'm' (Market price); On FX market – FX fixing price
				
				'o' (Official open price);
				'p' (Official current price);
				'q' (admitted quote); On FX market: international FX fixing price
				'r' (Official close price);
				'v' (Total bid volume);
				'w' (Total offer volume);
				
				's' (Dark pool Auction price)
				'x' (Dark Pool Auction volume), expressed in number of securities
				y’(Accrued coupon yield per the unit of security at current date, expressed in rubles)
				'u' (Duration);
				'z' (Trade list).
					trading boards for this Symbol (market, current, WAP prices, etc.)
					• The set of field values may be extended following the Trading system updates. It is recommended to allow in your code ignoring unknown values of this field, and linked to such entry values of other fields, until the new field meaning can be supported by your application.
					• Indexes are published in Market statistics (MSR and MSS) channels.
					• Preious trading day values are indicated by additional tag 286

=> 278	MDEntryID	N	String		Unique Market Data Entry identifier. Notes: • For trades (269=z) entries, contains a string with Exchange trade number that is equal to trade numbers in all trading interfaces • For aggregated orderbook (OBS and OBR channels) contains a unique string identifier of price level • For OrderList (OLR, OLS channels), contains a

					string identifier of Add Order (279=0) update for an order, NOT directly tied to the Exchange Order number in trading interfaces.
=> 270	MDEntryPx	C	Price		Price of the Market Data Entry. Conditionnally required if MDEntryType (269) not in ('A', 'B', 'C', 'J'). Conditionally required when MDEntryType = "auction clearing price"
=> 271	MDEntrySize	C	Qty		Quantity represented by the Market Data Entry. Conditionally required if MDEntryType (269) in ('0', '1', '2',’A’, 'B', 'C'). Conditionally required when MDEntryType =’Q’ (auction clearing price),‘g’(Offer volume market order in closing auction) Note: For 269=B, this field value is expressed in number of securities. For all other values of tag 269, this field value is expressed in number of lots.
=> 272	MDEntryDate	N	UTCDateOnly		Date of Market Data Entry.
=> 273	MDEntryTime	N	UTCTimeOnly		Time of Market Data Entry.
=> 336	TradingSessionID	N	String		Identifier for Trading Session. Used to represent SECBOARD. Note: an instrument with a given SecCode may be traded in several trading boards (SecBoard). You should use each Symbol (55)+TradingSessionID (336) combination as an individual security with own order book and list of trades.
=>625	TradingSessionSubID	N	String	NA – No trading O – Opening auction period C – Closing period F – Final closing period N – Normal trading period L – Closing auction period I – Discrete auction period D – Dark pool auction period E – Trading at the closing auction price period	Indicates the trading period For updates and snapshots, Period value indicates a period for an event reported, not necessarily the currently running period.
=> 276	QuoteCondition	N	MultipleValueString	'C' (Exchange Best)	Space-delimited list of conditions describing a quote.
=> 277	TradeConditi	N	MultipleValueStri	'C' (Cash Trade (same day clearing));	Space-delimited list of conditions describing a trade.

	on		ng	'J' (Next Day Trade (next day clearing));
				'R' (Opening Price) ;
				'AJ' (Official Closing Price);
				‘98’ (Minimum value);
				‘99’ (Maximum value).	

=> 286	OpenCloseSettlFlag	N	MultipleValueString	'4' (Entry from previous business day)	Flag that identifies a market data entry.
=>40	OrdType	H	Char	‘1’(Market)	Order type. Used when MDEntryType (269) =’g’,’f’ Note: Market in Closing Auction orders are activated and published in Order List feed in Closing Auction period. Matching occurs at the end of closing auction. Other market orders are not published in the feed because they never stay active.
=> 236	Yield	N	Percentage		Yield percentage.
=> 64	*SettlDate*	*N**	*LocalMktDate*		*Specific date of trade settlement (SettlementDate) in YYYYMMDD format* *Notes:* For trades – settlement date of regular trade or negotiated deal. For REPO trades – settlement date of first part of REPO.
=> 44	Price	N	Price		REPO rate for REPO trades.
=> 423	PriceType	N	int	‘1’ percentage	Indicates price type (REPO rate in percentage) for REPO trades.
=>5292	BidMarketSize	N	Int		Total volume of market buy orders calculated for currently expected auction price, expressed in number of securities. Used in closing auctions
=>5293	AskMarketSize	N	Int		Total volume of market sell orders, expressed in number of securitiesUsed in closing auctions.
=> 5384	AccruedInterestAmt	N	Amt		Amount of accrued interest.
=> 5459	*SettlType*	N	*Char*		*MOEX settlement code for trades (269=z)*
=> 5510	ChgFromWAPrice	N	PriceOffset		Indicates change from previous day's weighted average price vs. last traded price.
=>5558	BuyBackPx	N	Price		For REPO deals - REPO value calculated in roubles for the current date (used in Trade List (269=z) feed).

=>5559	BuyBackDate	N	LocalMktDate		For REPO deals - the date of the second part of REPO (used in Trade List (269=z) feed). Published as REPO buyback duration REPOTERM+<Settledate>
=>5677	Repo2Px	N	Price		Value of the 2nd (buy-back) REPO leg, expressed in settlement currency (used in Trade List (269=z) feed).
=>5791	TotalVolume	H	Amt		Total volume. Used when MDEntryType (269)='f' Market in auction buy orders have money volume instead of lot quantity. Other orders use lot quantities.
=> 5902	EffectiveTime	N	UTSTimestamp		Order activation time. The order or price level with an activation time specified is not active until that time.
=>9820	StartTime	N	UTSTimestamp		Auction start time. Used for Dark pool and Discrete auctions
=> 6139	TotalNumOfTrades	N	int		Total number of trades.
=> 6143	TradeValue	N	Amt		Trade Value.
=>7017	VolumeIndicator	N	int	'0' (No orders) '1' (Total orders value is less than N*) '2' (Total orders value is greater than N*)	Volume indicator of Dark Pool auction active orders. Used when MDEntryType(269)='v' or 'w'. N(variable)*- the large order volume factor as determined by the Exchange .
=> 9168	OfferNbOr	N	int		Number of sell orders.
=> 9169	BidNbOr	N	int		Number of buy orders.
=>9280	NominalValue	N	float		In REPO with CCP trading boards (currently EQRP), participants do anonymous trading for REPO rate as a cost of money. For this trading mode, underlying securities prices in main market are discounted for REPO trading based on CCP (Central Counter Party) risk management parameters. Discounts may depend on individual order size. For each order, the system calculates its money value based on underlying security's discounted price and number of lots in the order.

					These amounts are then aggregated in the orderbook at each REPO rate level and published in the REPO with CCP aggregated orderbook as an additional field 9280. This field is used in OBR/OBS channels only for REPO with CCP on-book trading.
=> 9412	OrigTime	N	int		Indicates the microseconds portion of the transaction's registration time at the Matching engine. Should be added to tag's 273 value to get microsecond precision timestamp. The field is available in Orders and trades channels.
=> 10504	OrderSide	N	char		Side of order.
=> 10505	OrderStatus	N	char	'O' (Active); 'T' (Order activation time hasn't come yet).	Describes the current state of order. Orders in T status are not active and not used in matching.
=>10509	MinCurrPx	N	Price		Minimum current price. Used to determine condition when the short sales should be prohibited.
=>10510	MinCurrPxChgTime	N	UTCTimeOnly		Time when minimum current price was changed.

4.3.6 Market Data - Incremental Refresh (X)

Important processing notes:

- Publishing massive updates in traffic-shaped feeds takes some time. At trading period end or start, this publishing is also done in parallel with publishing massive Security Status (35=f) messages. Parallel publishing may result in getting an incremental update from new trading state slightly before receiving the status change for a security, or getting an incremental update from previous trading state after the trading status changes to new trading period.
- For channels where add, change and delete MDUpdateActions are possible (Orders, Orderbook) the correct state is achieved after processing the whole set of repeating group entries in the message.
- FAST message length is limited by the network MTU size, current limitation is 1300 bytes. For massive updates, this results in splitting the data per several FAST messages. In this case, it is recommended to continue processing messages until you receive an update with FAST message size well less than the maximum length. Otherwise you may get short time crossed book state.
- There is no Delete or Change actions for Trades feed.

Table 15

Tag	Field name	Req'd	Type	Valid values	Comments
<Standard Message Header>		Y			MsgType = 'X'
268	NoMDEntries	Y	NumInGroup		Number of entries in Market Data message.

=> 279	MDUpdateAction	Y	char	'0' (New);
				'1' (Change);
				'2' (Delete).
					Type of Market Data update action.

=> 269	MDEntryType	C	char	'0' (Bid);
				'1' (Offer);
				'2' (Last Trade in Market statistics feed);
				'3' (Index Value);
				'4' (Opening Price);
				'5' (Closing Price);
				'7' (Trading Session High Price);
				'8' (Trading Session Low Price);
				'9' (Trading Session VWAP Price);
				'A' (Imbalance), expressed in number of securities
				'B' (Trade Volume), expressed in number of securities;
				'J' (Empty book);
				'N' (Session high bid);
				'O' (Session low offer);
				'Q' (Closing Auction clearing price); the clearing volume (271) is expressed in lots;
				'W'(Closing auction price);
				'c'(Closing auction volume), expressed in number of securities ;
				'f' (For MSR/MSS feeds - volume of buy market orders in closing auction, expressed in number of securities; for OLR/OLS – market in closing auction buy order);
				'g'(For MSR/MSS feeds: volume of sell market orders in closing auction, expressed in number of securities; for OLR/OLS – market in closing auction sell order);
				'i' (Last bid price);
				'j' (Last offer price);
				'h' (Open period price);
				'k' (Close period price);
					Type Market Data entry.
					Notes:
					• The availability of this field's values depends on market type (FX or Equities), SecBoard code (336) and the Exchange trading rules.
					• Different feeds have subsets of possible values, depending on the data contents.
					• Empty Book (269=J) indicates no data for a security. Empty Book message may be generated market-wide, which indicates that you should remove all previously collected data and start over.
					• Meaning of some values depend on market type (FX or Equities) and corresponding trading rules
					• Off-book trading boards do not have data in Orderbook Snapshot (OBS) and OrderList snapshot feeds (OLS).
					• Off-book trading boards may have market statistics data for a Symbol taken from on-book trading boards for this Symbol (market, current, WAP prices, etc.)
					• The set of field values may be extended following the Trading system updates. It is recommended to allow in your code ignoring unknown values of this field, and linked to such entry values of other fields, until the new field meaning can be supported by your application.
					• Indexes are published in Market statistics (MSR and MSS) channels.
					• Preious trading day values are indicated by

				'l' (Market price 2); on FX market – FX fixing price as calculated between 11:59 and 12:00 Moscow time. 'm' (Market price); On FX market – FX fixing price 'o' (Official open price); 'p' (Official current price); *'q' (Last admitted quote); On FX market: international FX fixing price* 'r' (Official close price); 'v' (Total bid volume); 'w' (Total offer volume); *'s' (Dark pool Auction price)* *'x' (Dark Pool Auction volume), expressed in number of securities.;* *y'(Accrued coupon yield per the unit of security at current date, expressed in rubles);* 'u' (Duration); 'z' (Trade list).	additional tag 286
=> 278	MDEntryID	N	String		Unique Market Data Entry identifier. Used, for example, for TRADENO. Notes: • For trades (269=z) entries, contains a string with Exchange trade number that is equal to trade numbers in all trading interfaces • For aggregated orderbook (OBS and OBR channels) contains a unique string identifier of price level • For OrderList (OLR, OLS channels), contains a string identifier of Add Order (279=0) update for an order, NOT directly tied to the Exchange Order number in trading interfaces.
=> 55	Symbol	Y	String		Ticker symbol. The MOEX internal instrument identifier, SecCode. Note: an instrument with a given SecCode may be traded in several trading boards (SecBoard). You should use each Symbol (55)+TradingSessionID (336) combination as an individual security with own order book and list of trades.
=> 83	RptSeq	Y	int		Sequence number of message within report series. Incremented by one for each update entry and for security status updates.
=> 270	MDEntryPx	C	Price		Price of the Market Data Entry.

					Conditionally required when MDUpdateAction (279) = New(0) and MDEntryType (269) not in ('A', 'B', 'C', *'J'*). Conditionally required when MDEntryType (269) = "Auction Clearing Price"
=> 271	MDEntrySize	C	Qty		Quantity represented by the Market Data Entry. Conditionally required when MDUpdateAction = New(0) and MDEntryType (269) in ('0', '1', '2', ‘A’, 'B', 'C'). Conditionally required when MDEntryType = ‘Q’ (auction clearing price), ‘g’(Offer volume market order in closing auction) Note: For 269=B, this field value is expressed in number of securities. For all other values of tag 269, this field value is expressed in number of lots.
=> 272	MDEntryDate	N	UTCDateOnly		Date of Market Data Entry.
=> 273	MDEntryTime	N	UTCTimeOnly		Time of Market Data Entry.
=> 336	TradingSessionID	N	String		Identifier for Trading Session. Used to represent SECBOARD. Note: an instrument with a given SecCode may be traded in several trading boards (SecBoard). You should use each Symbol (55)+TradingSessionID (336) combination as an individual security with own order book and list of trades.
=> 625	TradingSessionSubID	N	String	NA – No trading O – Opening auction period C – Closing period F – Final closing period N – Normal trading period L – Closing auction period I – Discrete auction period D – Dark pool auction period E – Trading at the closing auction price period	Indicates the trading period For updates and snapshots, Period value indicates a period for an event reported, not necessarily the currently running period.
=> 276	QuoteCondition	N	MultipleValueString	'C' (Exchange Best)	Space-delimited list of conditions describing a quote.

=> 277	TradeCondition	N	MultipleValueString	'C' (Cash Trade (same day clearing));
				'J' (Next Day Trade (next day clearing));
				'R' (Opening Price) ;
				'AJ' (Official Closing Price);
				‘98’ (Minimum value);
				‘99’ (Maximum value).
					Space-delimited list of conditions describing a trade.

=> 286	OpenCloseSettlFlag	N	MultipleValueString	'4' (Entry from previous business day)	Flag that identifies a market data entry.
=> 40	OrdType	H	Char	‘1’(Market)	Order type. Used when MDEntryType (269) =’g’,’f’ Note: Market in Closing Auction orders are activated and published in Order List feed in Closing Auction period. Matching occurs at the end of closing auction. Other market orders are not published in the feed because they never stay active.
=> 451	NetChgPrevDay	N	PriceOffset		Net change from previous day closing price vs. last traded price.
=> 236	Yield	N	Percentage		Yield percentage.
=> 64	*SettlDate*	*N**	*LocalMktDate*		*Specific date of trade settlement (SettlementDate) in YYYYMMDD format* *Notes:* For trades – settlement date of regular trade or negotiated deal. For REPO trades – settlement date of first part of REPO.
=> 44	Price	N	Price		REPO rate for REPO trades
=> 423	PriceType	N	int	‘1’ percentage	Indicates price type (REPO rate in percentage) for REPO trades.
=> 5292	BidMarketSize	N	Int		Total volume of market buy orders calculated for currently expected auction price, expressed in number of securities.Used in closing auctions
=> 5293	AskMarketSize	N	Int		Total volume of market sell orders, expressed in number of securitiesUsed in closing auctions
=> 5384	AccruedInterestAmt	N	Amt		Amount of accrued interest.
=> 5459	*SettlType*	N	*Char*		*MOEX settlement code for trades (269=z)*
=> 5510	ChgFromWAPrice	N	PriceOffset		Indicates change from previous day's weighted average price vs. last traded price.
=> 5558	BuyBackPx	N	Price		For REPO deals - REPO value calculated in roubles for the current date (used in Trade List (269=z) feed).
=> 5559	BuyBackDate	N	LocalMktDate		For REPO deals - the date of the second part of REPO (used in Trade List (269=z) feed). Published as REPO buyback duration REPOTERM+<Settledate>

=> 5677	Repo2Px	N	Price		Value of the 2nd (buy-back) REPO leg, expressed in roubles (used in Trade List (269=z) feed).
=> 5791	TotalVolume	H	Amt		Used when MDEntryType (269)=’f’ Market in auction buy orders have money volume instead of lot quantity. Other orders use lot quantities.
=> 5902	EffectiveTime	N	UTSTimestamp		Order activation time. The order or price level with an activation time specified is not active until that time.
=> 9820	StartTime	N	UTSTimestamp		Auction start time. Used for Dark pool and Discrete auctions
=> 6139	TotalNumOfTrades	N	int		Total number of trades.
=> 6143	TradeValue	N	Amt		Trade Value.
=>7017	VolumeIndicator	N	int	'0' (No orders) '1' (Less then N* minimum order value) '2' (Greater then N* minimum order value)	Volume indicator of Dark Pool auction active orders. Used when MDEntryType(269)=’v’ or ‘w’. N(variable)*- the large order volume factor as determined by the Exchange .
=> 9168	OfferNbOr	N	int		Number of sell orders.
=> 9169	BidNbOr	N	int		Number of buy orders.
=>9280	NominalValue	N	float		In REPO with CCP trading boards (currently EQRP), participants do anonymous trading for REPO rate as a cost of money. For this trading mode, underlying securities prices in main market are discounted for REPO trading based on CCP (Central Counter Party) risk management parameters. Discounts may depend on individual order size. For each order, the system calculates its money value based on underlying security’s discounted price and number of lots in the order. These amounts are then aggregated in the orderbook at each REPO rate level and published in the REPO with CCP aggregated orderbook as an additional field 9280. This field is used in OBR/OBS channels only for REPO with CCP on-book trading.
=> 9412	OrigTime	N	int		Indicates the microseconds portion of the transaction’s

					registration time at the Matching engine. Should be added to tag’s 273 value to get microsecond precision timestamp. The field is available in Orders and Trades channels.
=> 10504	OrderSide	N	char		Side of order.
=> 10505	OrderStatus	N	char	'O' (Active); 'T' (Order activation time hasn't come yet).	Describes the current state of order. Orders in T status are not active and not used in matching.
=>10509	MinCurrPx	N	Price		Minimum current price. Used to determine condition when the short sales should be prohibited.
=>10510	MinCurrPx ChgTime	N	UTCTimeOnly		Time when minimum current price was changed.

5. Network Connectivity Guide

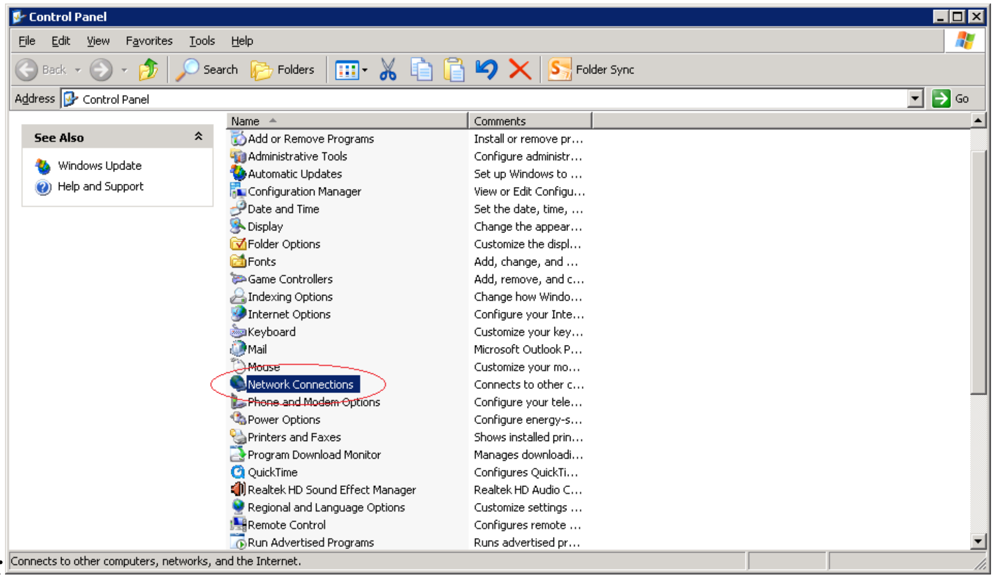
5.1. Configure a VPN connection with MOEX using Windows XP

To configure a VPN connection, do the following:

1. Make sure you are connected to the Internet;

2. Click *Start*, and then click *Control Panel;*

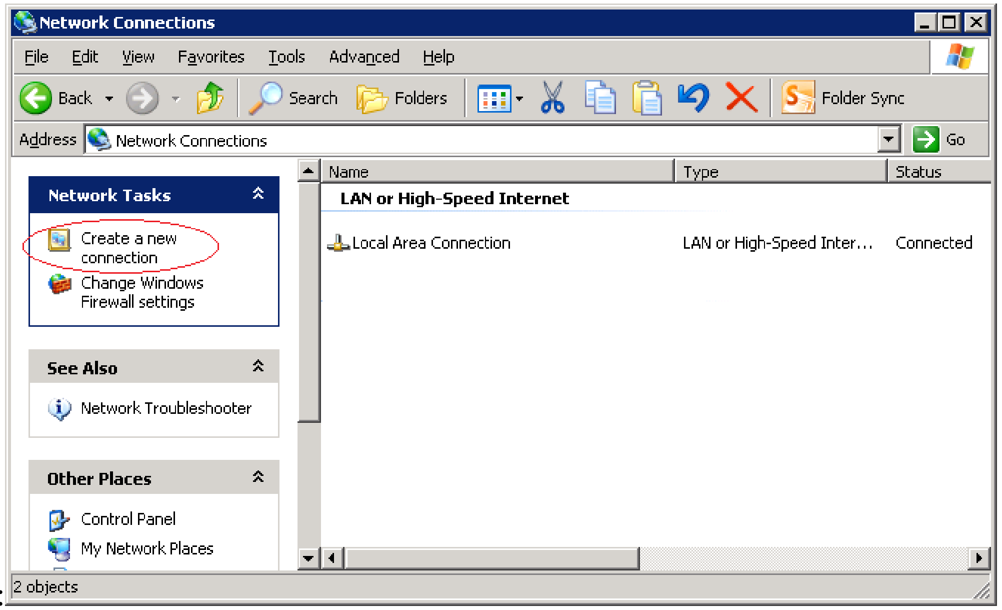
3. In *Control Panel*, double click *Network Connections:*



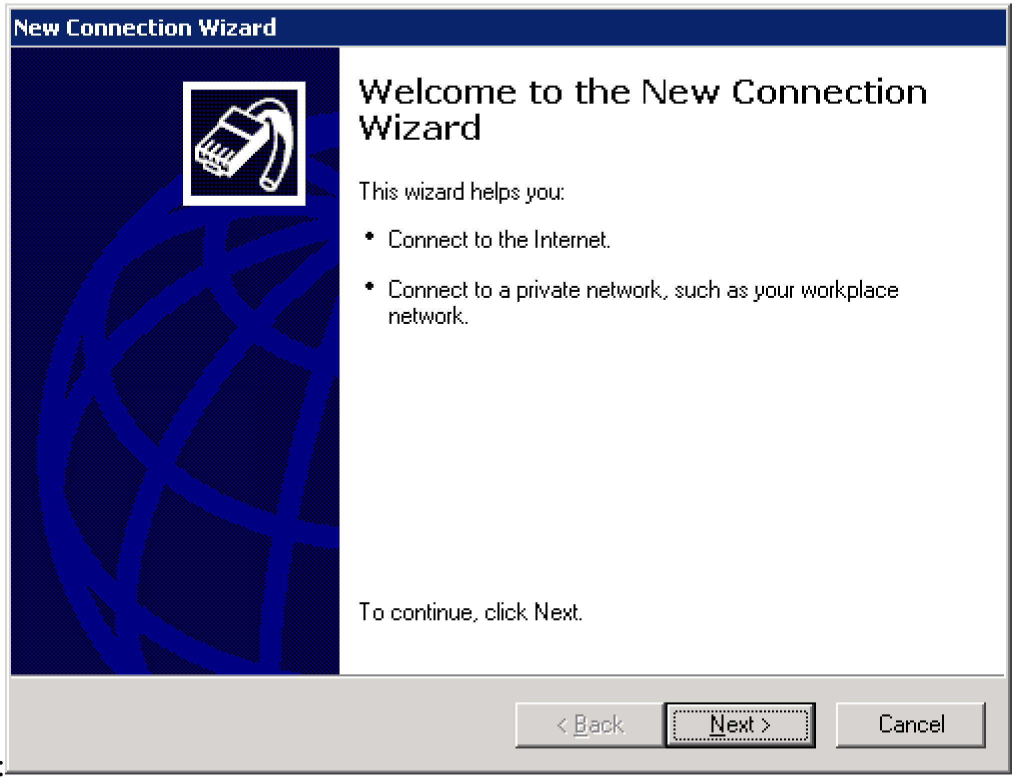
Name	Comments
Add or Remove Programs	Install or remove pr…
Administrative Tools	Configure administr…
Automatic Updates	Set up Windows to …
Configuration Manager	View or Edit Configu…
Date and Time	Set the date, time, …
Display	Change the appear…
Folder Options	Customize the displ…
Fonts	Add, change, and …
Game Controllers	Add, remove, and c…
Indexing Options	Change how Windo…
Internet Options	Configure your Inte…
Keyboard	Customize your key…
Mail	Microsoft Outlook P…
Mouse	Customize your mo…
Network Connections	Connects to other c…
Phone and Modem Options	Configure your tele…
Power Options	Configure energy-s…
Printers and Faxes	Shows installed prin…
Program Download Monitor	Manages downloadi…
QuickTime	Configures QuickTi…
Realtek HD Sound Effect Manager	Realtek HD Audio C…
Regional and Language Options	Customize settings …
Remote Control	Configures remote …
Run Advertised Programs	Runs advertised pr…

Connects to other computers, networks, and the Internet.

4. Click *Create a new connection* in the *Network Tasks* task pad:


Network Connections
File Edit View Favorites Tools Advanced Help
Back Search Folders Folder Sync
Address Network Connections Go
Network Tasks
Create a new connection
Change Windows Firewall settings
See Also
Network Troubleshooter
Other Places
Control Panel
My Network Places
Name Type Status
LAN or High-Speed Internet
Local Area Connection LAN or High-Speed Inter... Connected
2 objects

5. In the *Network Connection* Wizard, click *Next*:


New Connection Wizard

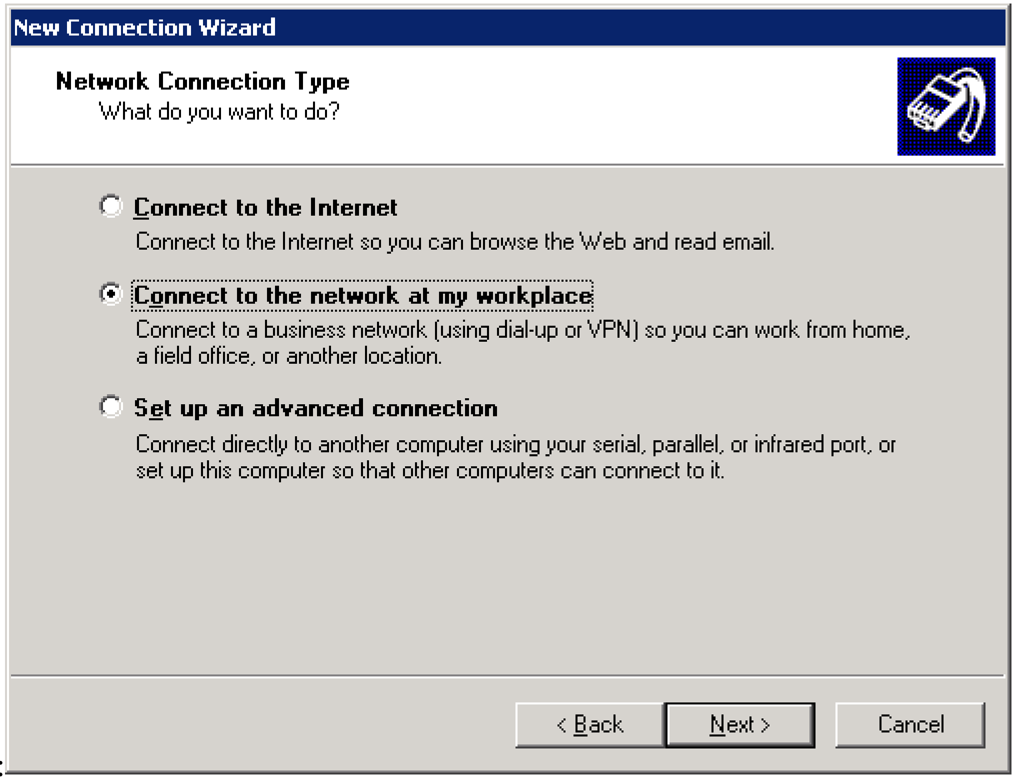
Welcome to the New Connection Wizard

This wizard helps you:

- Connect to the Internet.
- Connect to a private network, such as your workplace network.

To continue, click Next.

6. Click *Connect to the network at my workplace* and then *Next*:



New Connection Wizard

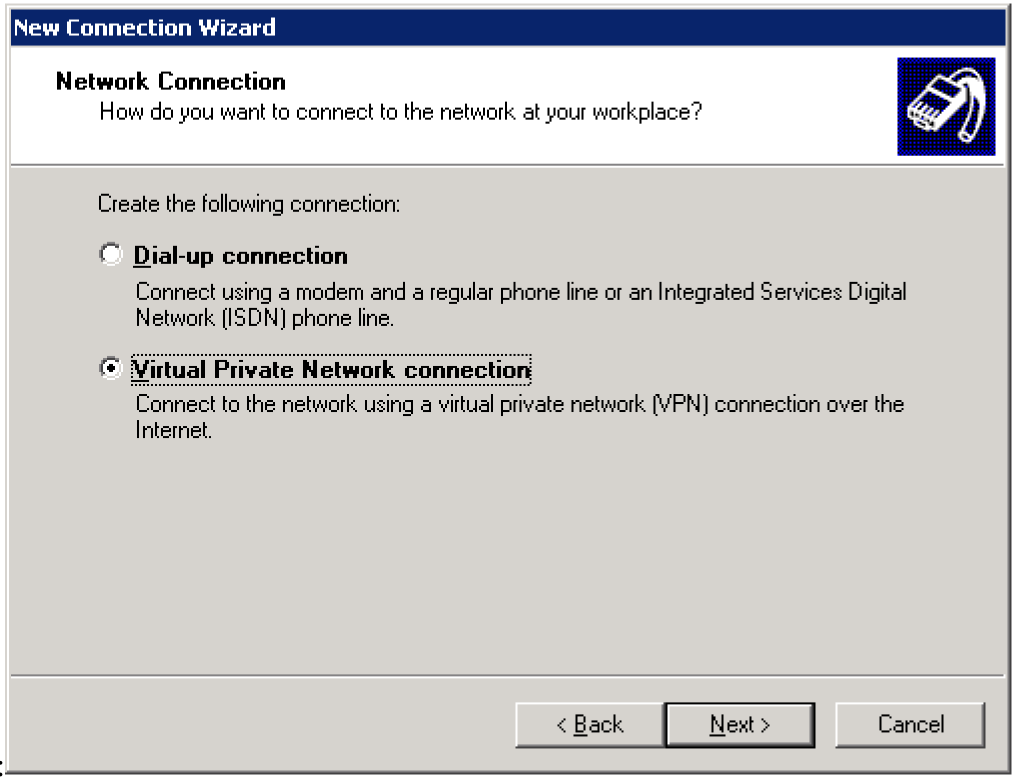
Network Connection Type
What do you want to do?

○ **Connect to the Internet**
Connect to the Internet so you can browse the Web and read email.

● **Connect to the network at my workplace**
Connect to a business network (using dial-up or VPN) so you can work from home, a field office, or another location.

○ **Set up an advanced connection**
Connect directly to another computer using your serial, parallel, or infrared port, or set up this computer so that other computers can connect to it.

7. Click *Virtual Private Network* connection and then *Next*:


New Connection Wizard

Network Connection

How do you want to connect to the network at your workplace?

Create the following connection:

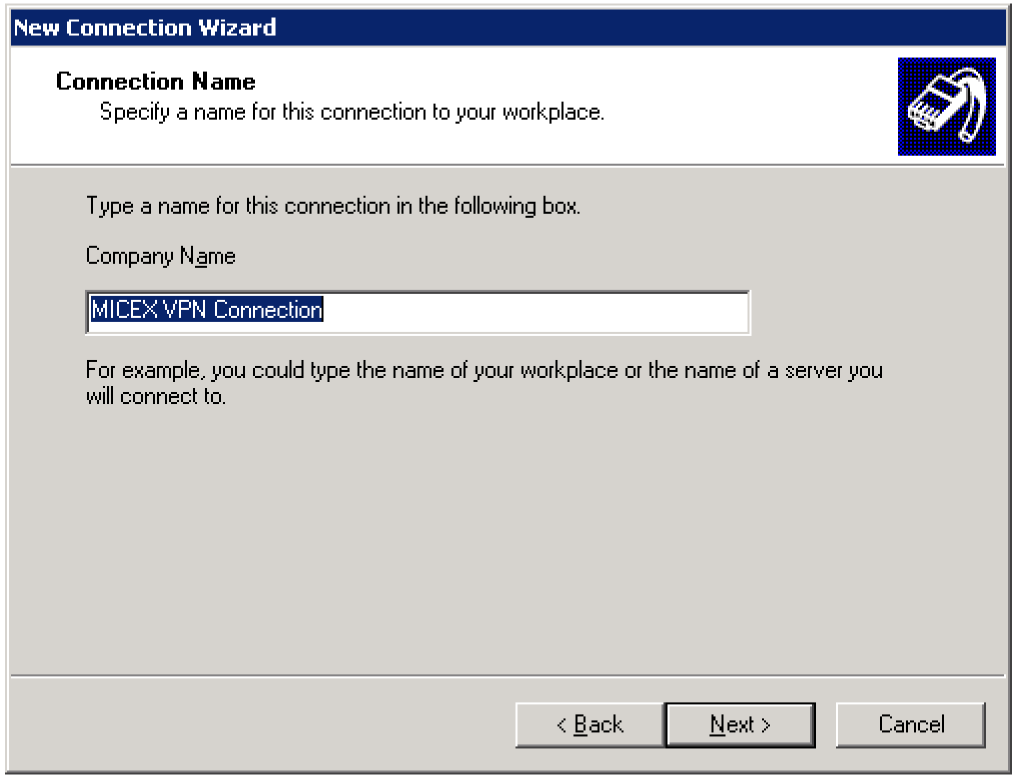
( ) **Dial-up connection**

Connect using a modem and a regular phone line or an Integrated Services Digital Network (ISDN) phone line.

[✓] **Virtual Private Network connection**

Connect to the network using a virtual private network (VPN) connection over the Internet.

8. Type *Company Name* (e.g. MOEX VPN Connection), and then click *Next*:


New Connection Wizard

Connection Name

Specify a name for this connection to your workplace.

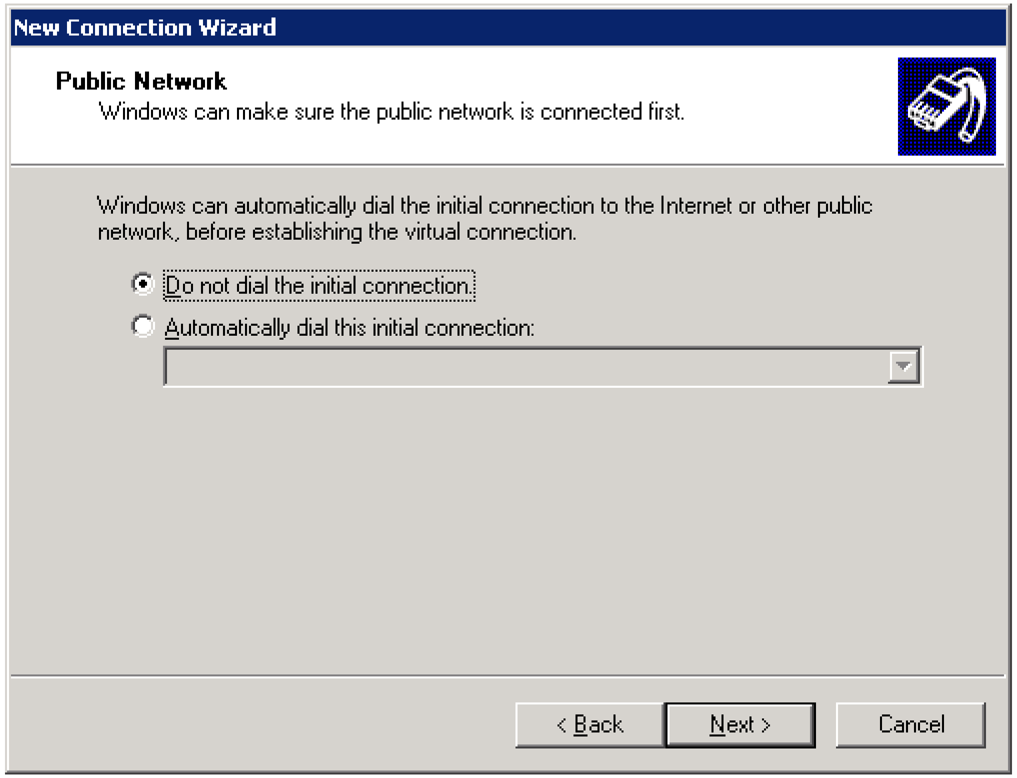
Type a name for this connection in the following box.

Company Name

MICEX VPN Connection

For example, you could type the name of your workplace or the name of a server you will connect to.

9. Click *Do not dial the initial connection*, and then click *Next*:


New Connection Wizard

Public Network

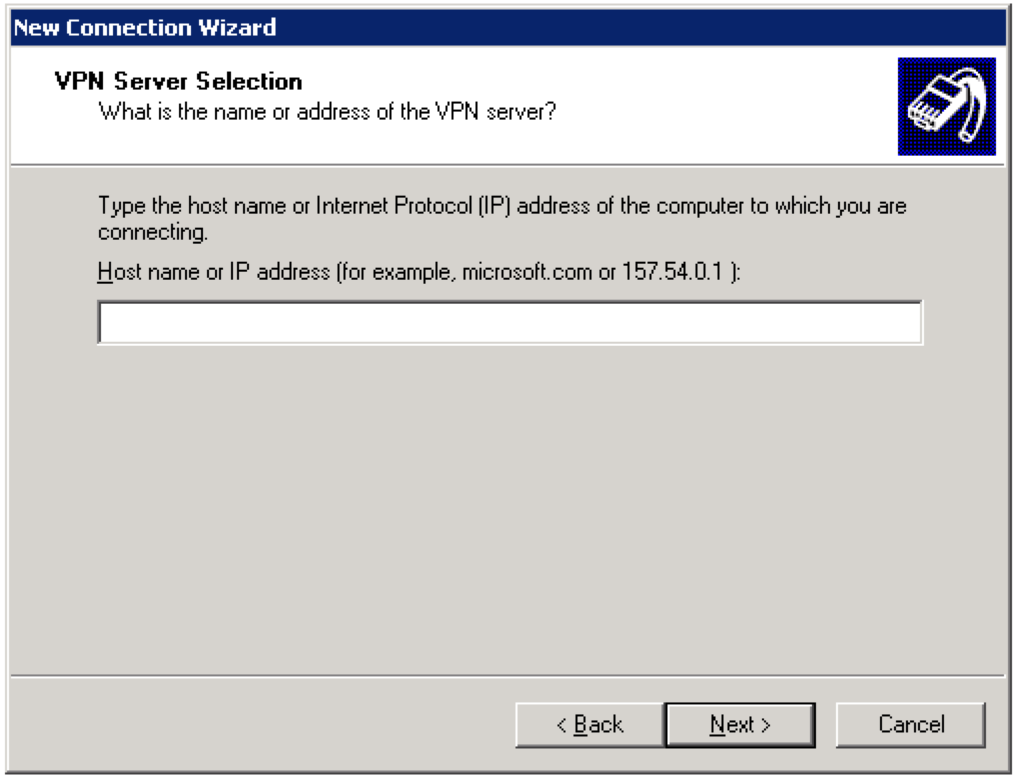
Windows can make sure the public network is connected first.

Windows can automatically dial the initial connection to the Internet or other public network, before establishing the virtual connection.

[● Do not dial the initial connection.]

○ Automatically dial this initial connection:

10. Type the server address provided by MOEX team, and then click *Next*:

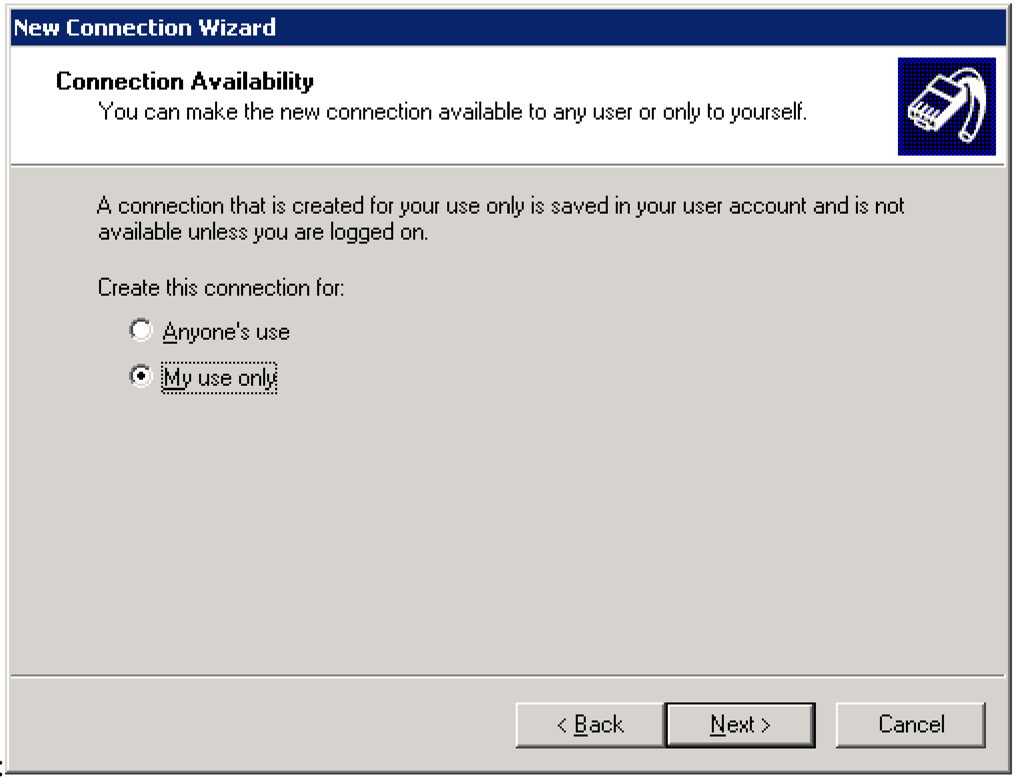


VPN Server Selection
What is the name or address of the VPN server?

Type the host name or Internet Protocol (IP) address of the computer to which you are connecting.

Host name or IP address (for example, microsoft.com or 157.54.0.1):

11. Click *My use only* and then *Next*:



New Connection Wizard

Connection Availability

You can make the new connection available to any user or only to yourself.

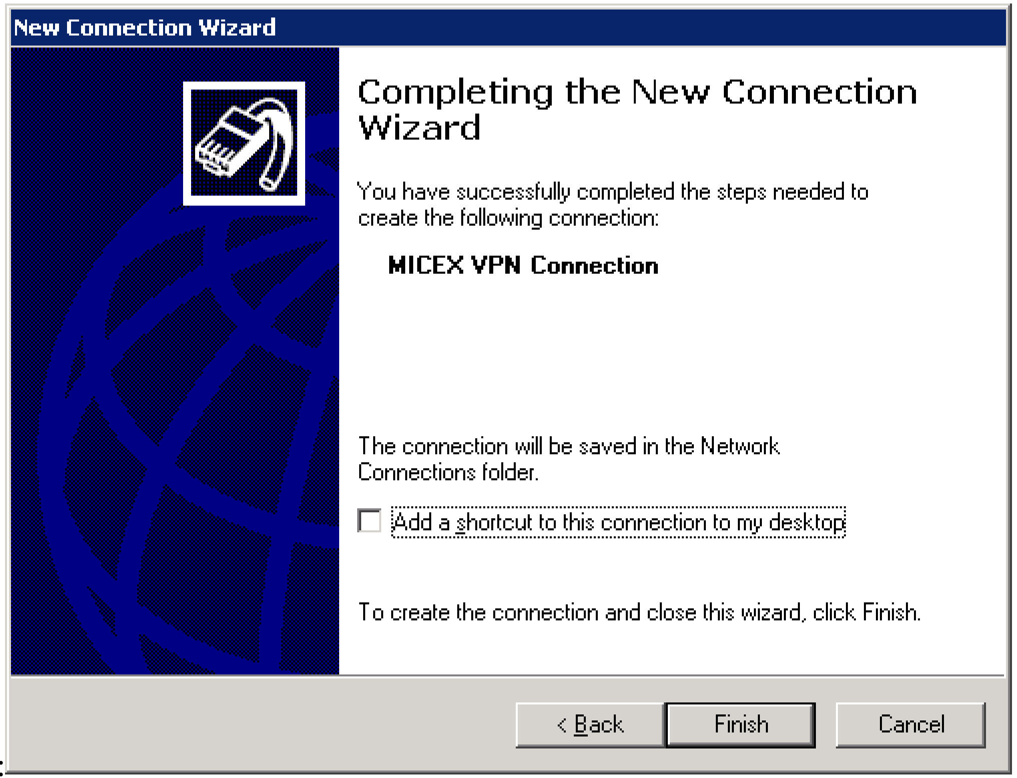
A connection that is created for your use only is saved in your user account and is not available unless you are logged on.

Create this connection for:

( ) Anyone's use

[✓ My use only]

12. Click *Finish*:



New Connection Wizard

Completing the New Connection Wizard

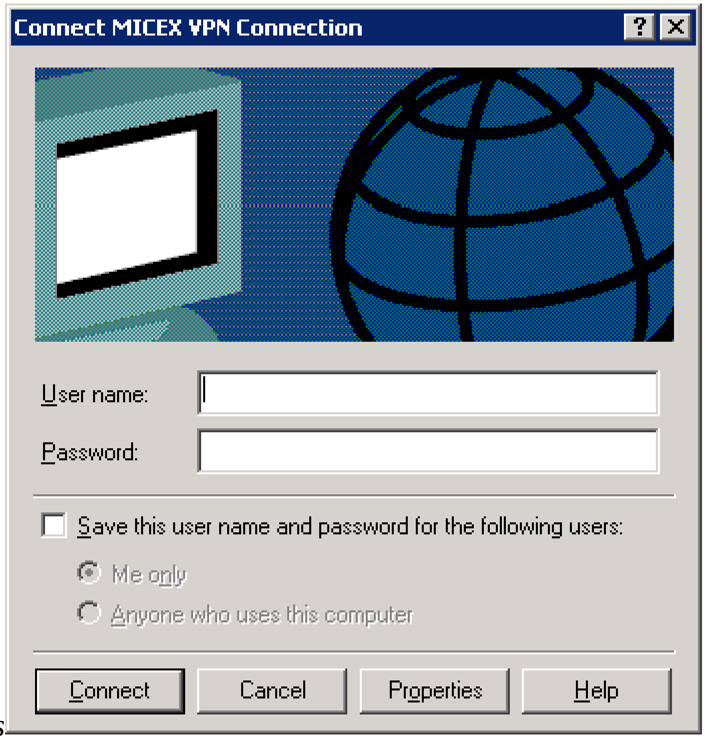
You have successfully completed the steps needed to create the following connection:

MICEX VPN Connection

The connection will be saved in the Network Connections folder.

☐ Add a shortcut to this connection to my desktop

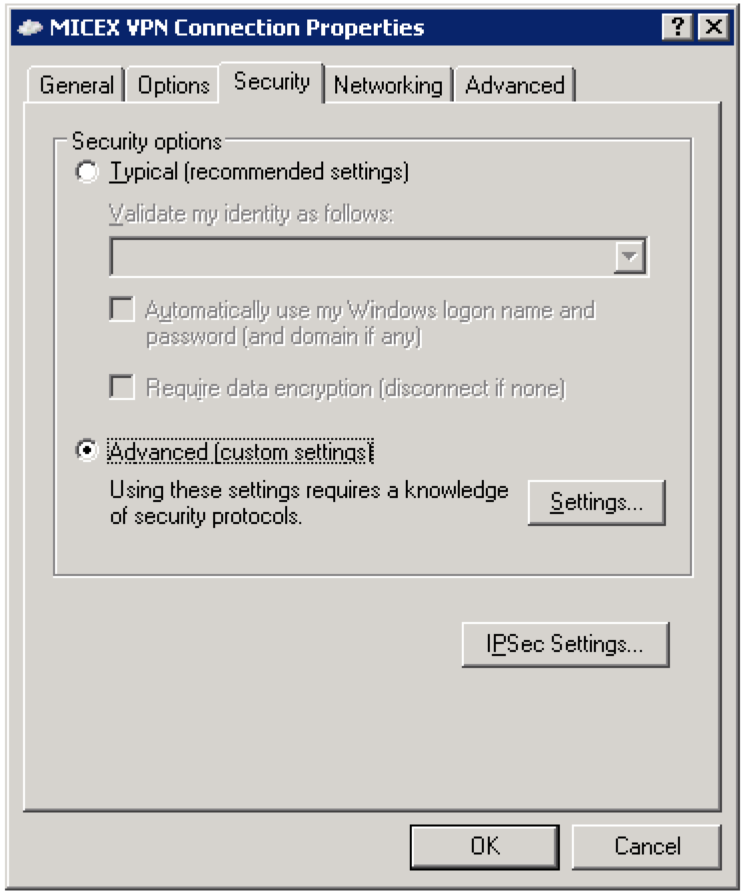
To create the connection and close this wizard, click Finish.



Connect MICEX VPN Connection
User name:
Password:
Save this user name and password for the following users:
Me only
Anyone who uses this computer
Connect
Cancel
Properties
Help

13. Leave *User name* and *Passwod* empty, and then click *Properties*

14. On *Security* tab, click *Advanced (custom settings)* and then *Settings…*:


MICEX VPN Connection Properties

Security options

( ) Typical (recommended settings)

Validate my identity as follows:

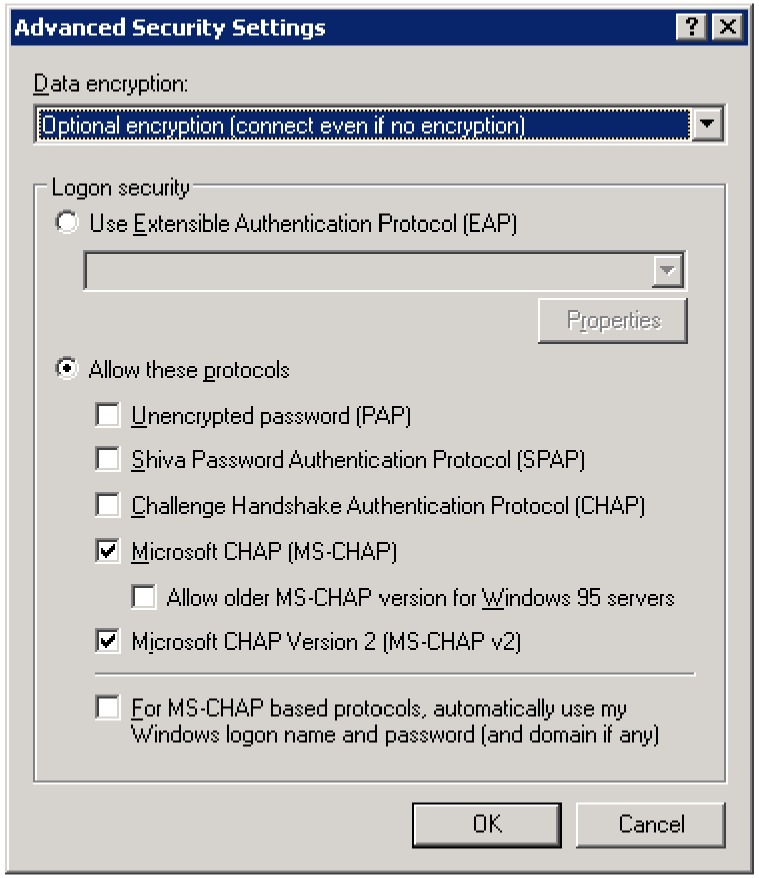
[ ] Automatically use my Windows logon name and password (and domain if any)

[ ] Require data encryption (disconnect if none)

(●) Advanced (custom settings)

Using these settings requires a knowledge of security protocols.

15. Choose *Optional encryption (connect even if no encryption)* data encryption and then click *OK*:



Advanced Security Settings

Data encryption:

Optional encryption (connect even if no encryption)

Logon security

( ) Use Extensible Authentication Protocol (EAP)

Properties

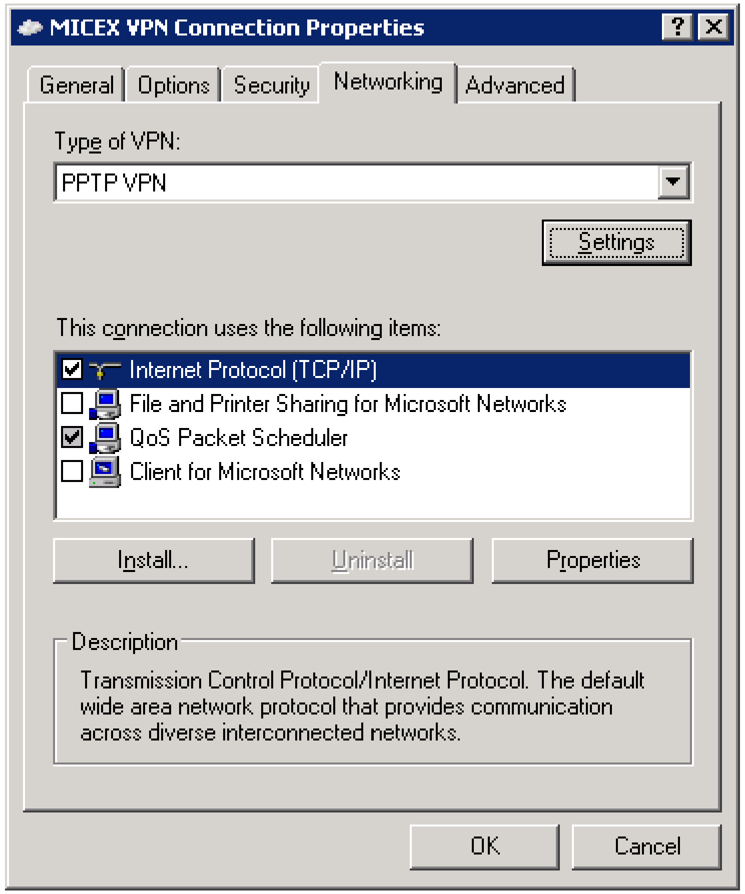
(●) Allow these protocols

- [ ] Unencrypted password (PAP)
- [ ] Shiva Password Authentication Protocol (SPAP)
- [ ] Challenge Handshake Authentication Protocol (CHAP)
- [x] Microsoft CHAP (MS-CHAP)
  - [ ] Allow older MS-CHAP version for Windows 95 servers
- [x] Microsoft CHAP Version 2 (MS-CHAP v2)

- [ ] For MS-CHAP based protocols, automatically use my Windows logon name and password (and domain if any)

OK Cancel

16. On *Networking* tab, choose *PPTP VPN* type of VPN and then click *OK*:


MICEX VPN Connection Properties

General | Options | Security | Networking | Advanced

Type of VPN:

PPTP VPN

Settings

This connection uses the following items:

- [✓] Internet Protocol (TCP/IP)
- [ ] File and Printer Sharing for Microsoft Networks
- [✓] QoS Packet Scheduler
- [ ] Client for Microsoft Networks

Install... | Uninstall | Properties

Description

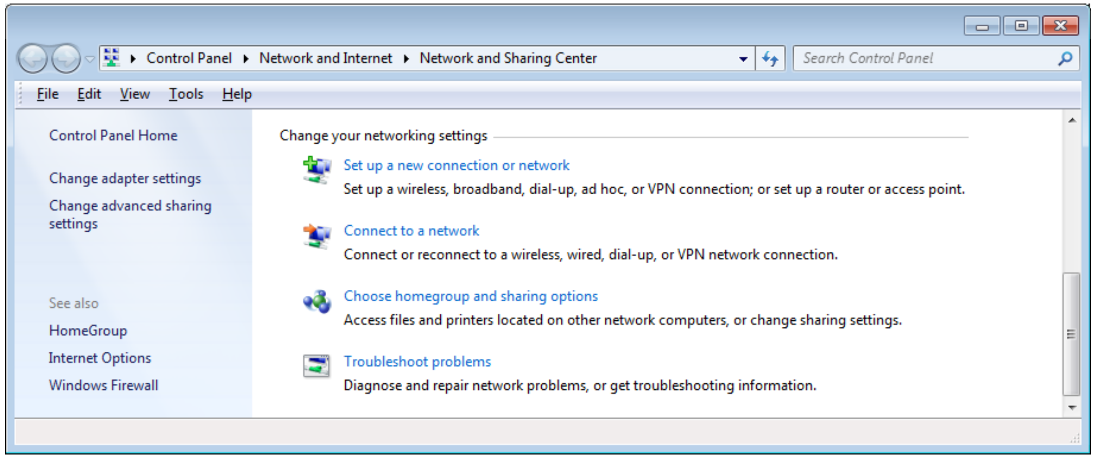
Transmission Control Protocol/Internet Protocol. The default wide area network protocol that provides communication across diverse interconnected networks.

OK | Cancel

5.2. Configure a VPN connection with MOEX using Windows 7

1. Make sure you are connected to the Internet

2. Open *Control Panel→Network and Internet→Network and Share Center* and then click *Set up a new connection or network*:


Control Panel ▸ Network and Internet ▸ Network and Sharing Center
Search Control Panel
File Edit View Tools Help
Control Panel Home
Change adapter settings
Change advanced sharing settings
See also
HomeGroup
Internet Options
Windows Firewall

Change your networking settings

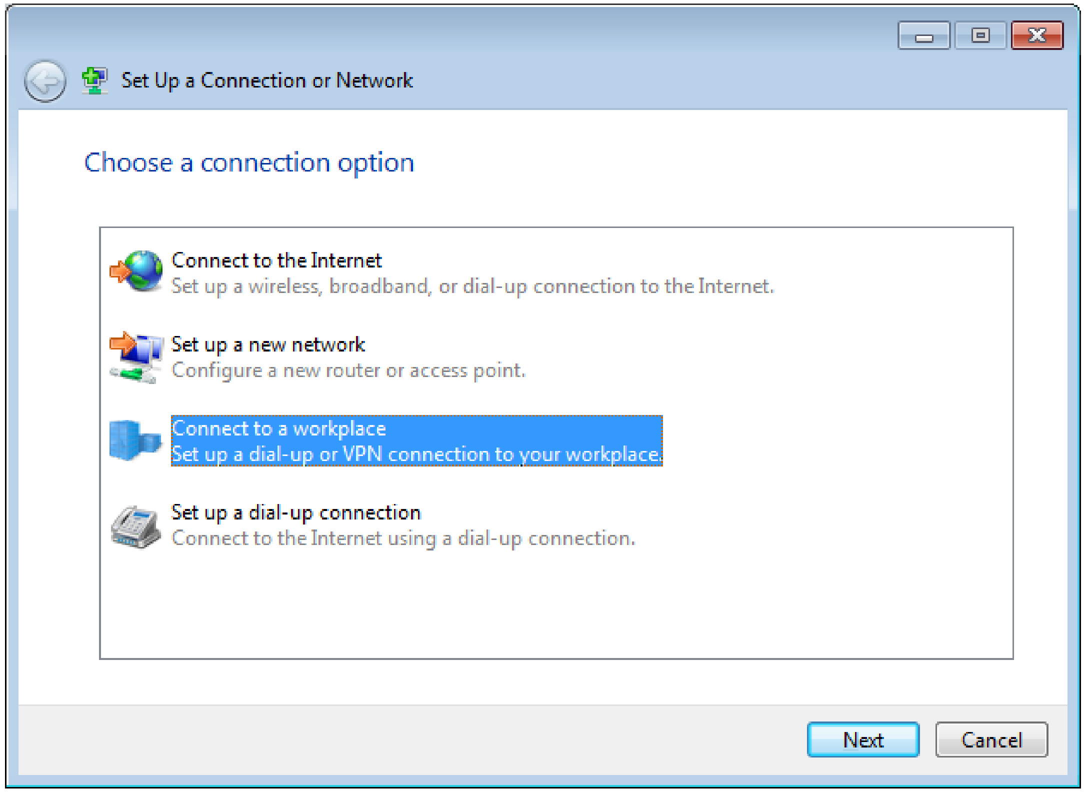
Set up a new connection or network
Set up a wireless, broadband, dial-up, ad hoc, or VPN connection; or set up a router or access point.

Connect to a network
Connect or reconnect to a wireless, wired, dial-up, or VPN network connection.

Choose homegroup and sharing options
Access files and printers located on other network computers, or change sharing settings.

Troubleshoot problems
Diagnose and repair network problems, or get troubleshooting information.

3. Choose *Connect to a workplace* and then click *OK*:


Set Up a Connection or Network

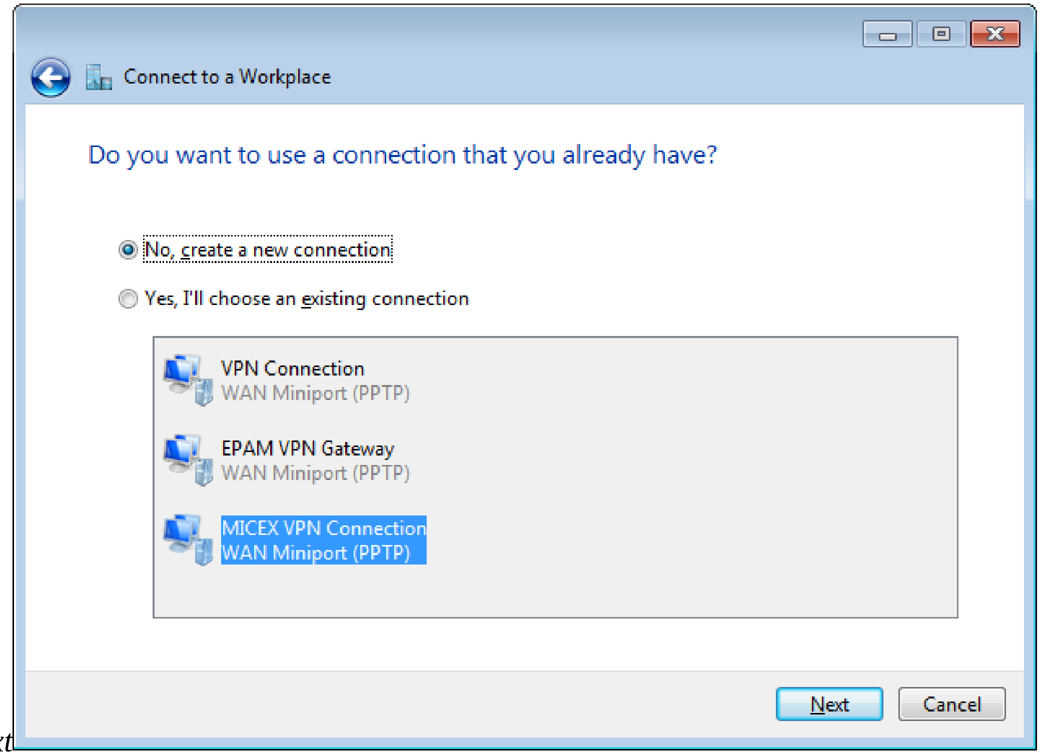
Choose a connection option

Connect to the Internet
Set up a wireless, broadband, or dial-up connection to the Internet.

Set up a new network
Configure a new router or access point.

Connect to a workplace
Set up a dial-up or VPN connection to your workplace.

Set up a dial-up connection
Connect to the Internet using a dial-up connection.


Connect to a Workplace

Do you want to use a connection that you already have?

[✓ No, create a new connection]

Yes, I'll choose an existing connection

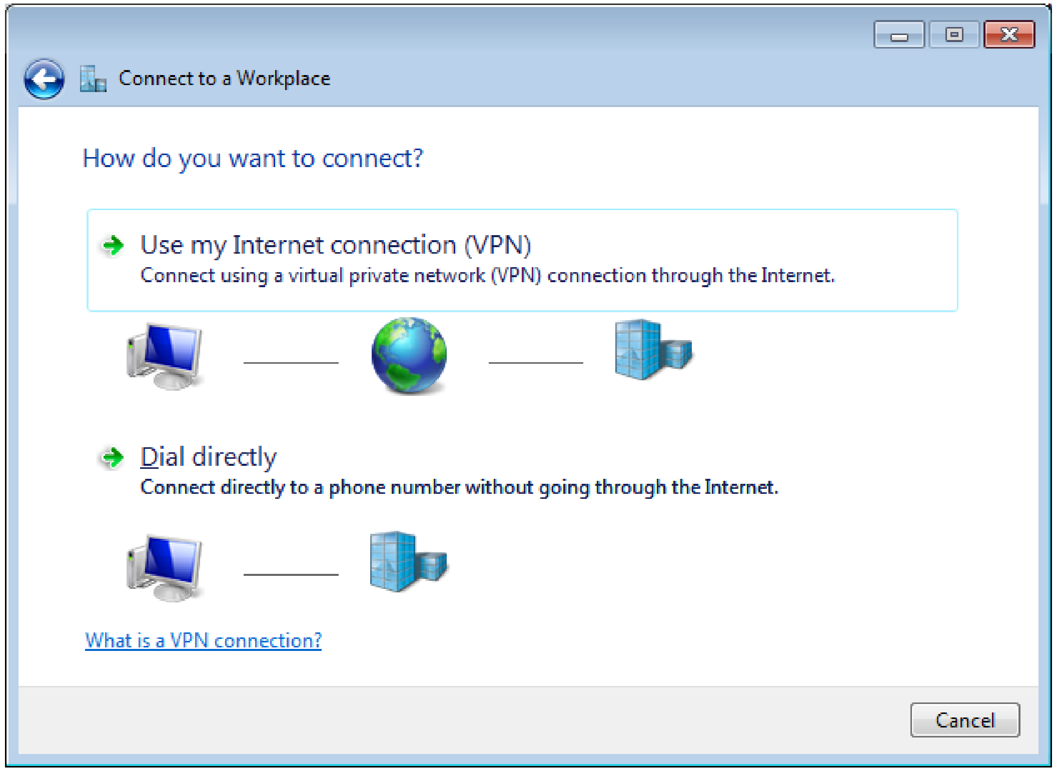
VPN Connection
WAN Miniport (PPTP)

EPAM VPN Gateway
WAN Miniport (PPTP)

MICEX VPN Connection
WAN Miniport (PPTP)

4. Choose *No, create a new connection* and then click *Next*

5. Click *Use my Internet Connection (VPN)*:



Connect to a Workplace

How do you want to connect?

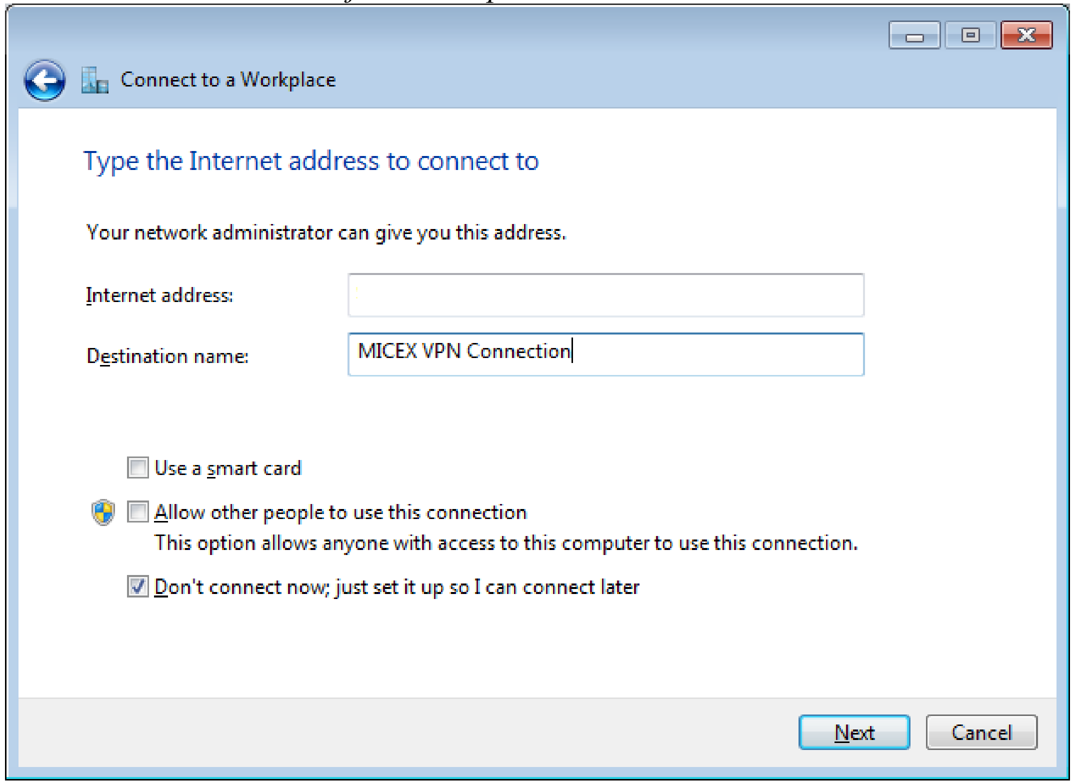
Use my Internet connection (VPN)
Connect using a virtual private network (VPN) connection through the Internet.

Dial directly
Connect directly to a phone number without going through the Internet.

What is a VPN connection?

Cancel

6. Type the server address provided by MOEX team to the *Internet address* field, type MOEX VPN Connection to the *Destination name* field, check *Don't connect now; just set it up so I can connect later* and then click *Next*:



Type the Internet address to connect to

Your network administrator can give you this address.

Internet address:

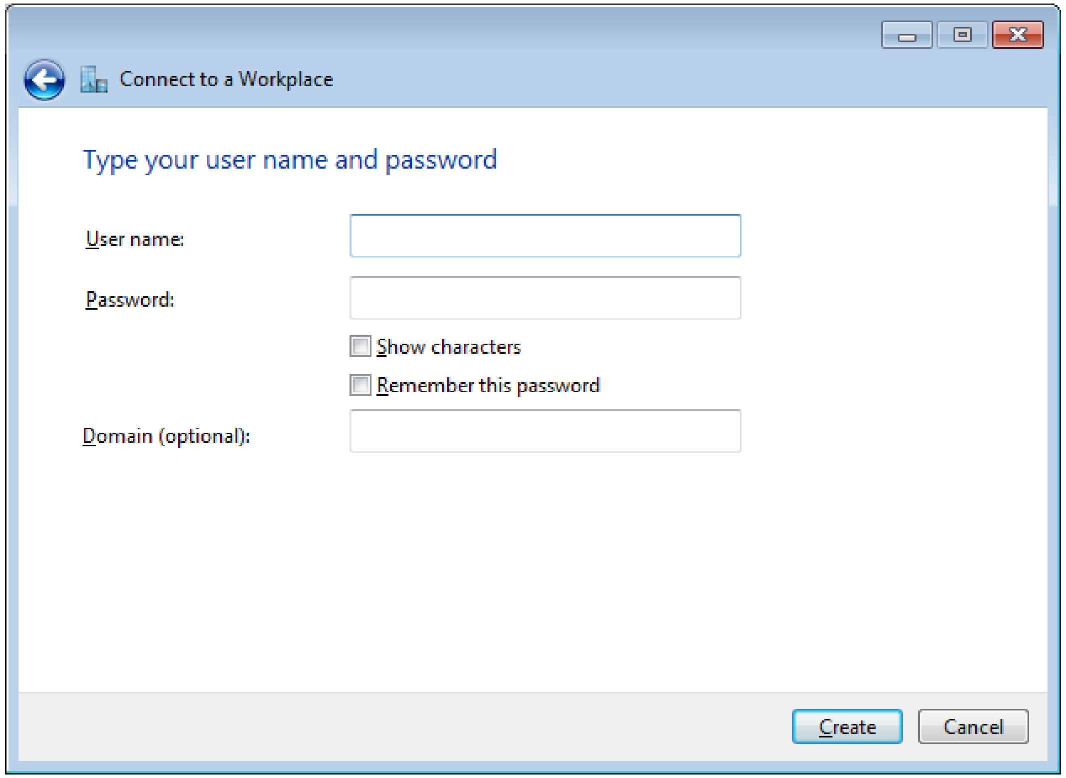
Destination name: MICEX VPN Connection

☐ Use a smart card

☐ Allow other people to use this connection
This option allows anyone with access to this computer to use this connection.

☑ Don't connect now; just set it up so I can connect later

7. Leave the next page without changes and then click *Next*:



Connect to a Workplace

Type your user name and password

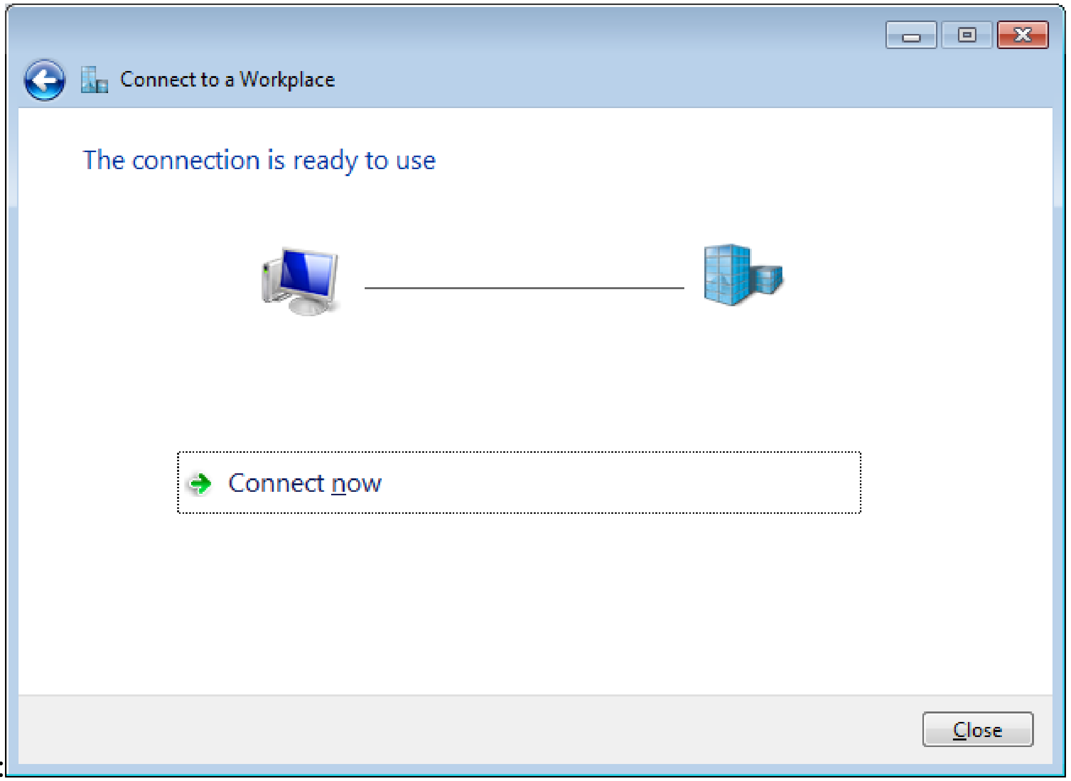
User name:

Password:

☐ Show characters

☐ Remember this password

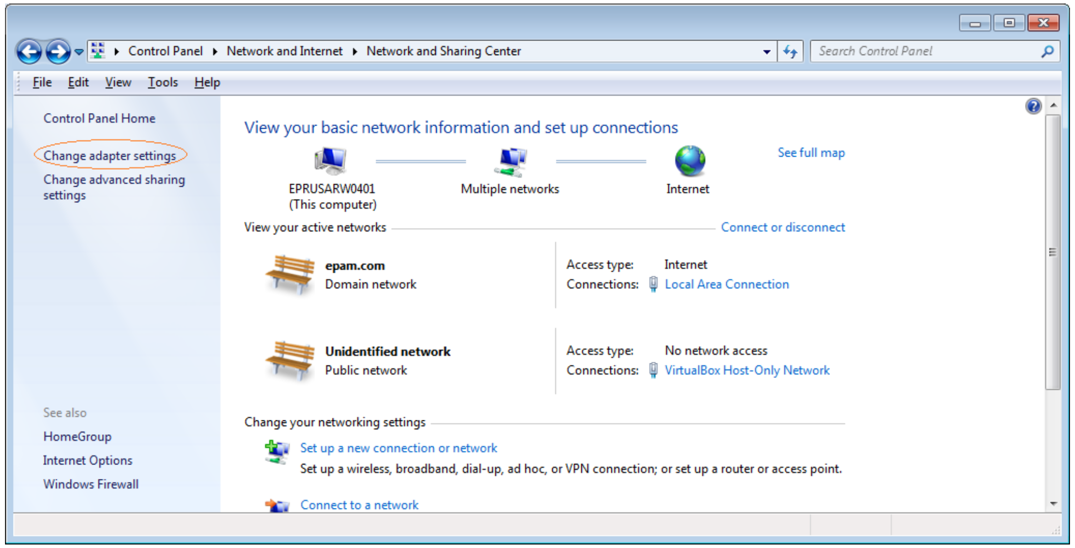
Domain (optional):


Connect to a Workplace

The connection is ready to use

8. Click *Close*:

9. Open *Control Panel→Network and Internet→Network and Share Center* and click *Change adapter setting*:



View your basic network information and set up connections

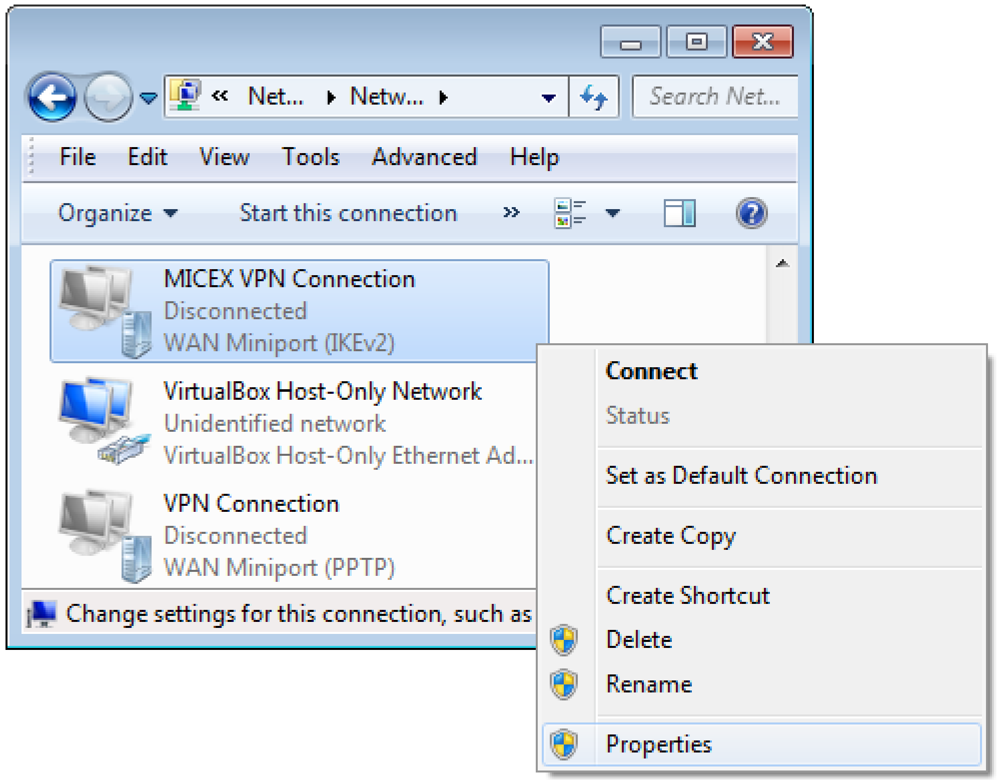
View your active networks

epam.com Domain network	Access type: Internet Connections: Local Area Connection
Unidentified network Public network	Access type: No network access Connections: VirtualBox Host-Only Network

Change your networking settings

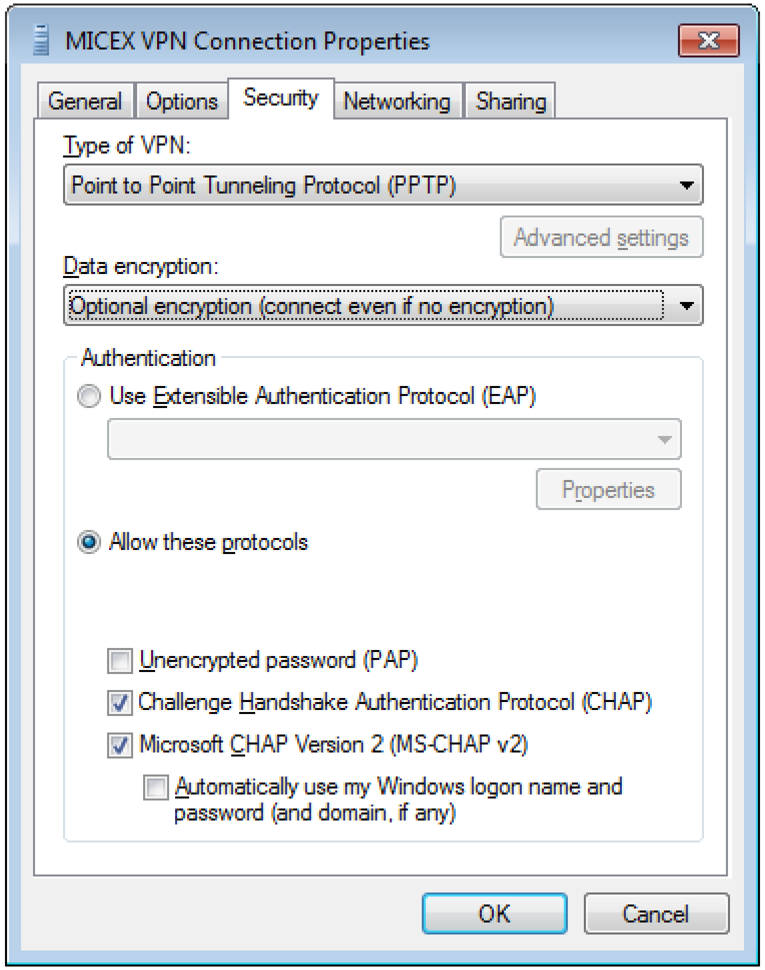
Set up a new connection or network
Set up a wireless, broadband, dial-up, ad hoc, or VPN connection; or set up a router or access point.

Connect to a network



10. Choose *Properties* of the just created connection:

11. On *Security* tab choose *Point to Point Tunneling Protocol (PPTP)* VPN type, choose *Optional encryption (connect even if no encryption)* data encryption and then click *OK*:


MICEX VPN Connection Properties

General | Options | Security | Networking | Sharing

Type of VPN:
Point to Point Tunneling Protocol (PPTP)

Advanced settings

Data encryption:
Optional encryption (connect even if no encryption)

Authentication
- ○ Use Extensible Authentication Protocol (EAP)

Properties

- ● Allow these protocols
  - ☐ Unencrypted password (PAP)
  - ☑ Challenge Handshake Authentication Protocol (CHAP)
  - ☑ Microsoft CHAP Version 2 (MS-CHAP v2)
    - ☐ Automatically use my Windows logon name and password (and domain, if any)

OK | Cancel

5.3. Configure a VPN connection with MOEX using OpenSUSE

1. Make sure you are connected to the Internet;

2. Install *pptp* client using the following command:

```
sudo zypper install pptp
```

3. Run the following command:

```
sudo /usr/sbin/pptp-command setup
```

4. Type ‘4’ and press enter:

```
1.) Manage CHAP secrets
2.) Manage PAP secrets
3.) List PPTP Tunnels
4.) Add a NEW PPTP Tunnel
5.) Delete a PPTP Tunnel
6.) Configure resolv.conf
7.) Select a default tunnel
8.) Quit
?: 4 + <enter>
```

5. Type ‘1’ and press enter:

```
Add a NEW PPTP Tunnel.

1.) Other
Which configuration would you like to use?: 1 + <enter>
```

6. Type `‘micex_vpn_connection’` and press enter:

```
Tunnel Name: micex_vpn_connection + <enter>
```

7. Type `‘<server address>‘` and press enter:

```
Server IP: <server address> + <enter>
```

8. Type ‘`del default`’ and press enter:

```
route: del default + <enter>
```

9. Type `‘add default gw 1.1.1.1 TUNNEL_DEV’` and press enter:

```
route: add default gw 1.1.1.1 TUNNEL_DEV
```

10. Simply press enter:

```
route: <enter>
```

11. Type ‘test’ and press enter:

```
Local Name: test
```

12. Leave a default value, simply press enter:

```
Remote Name [PPTP]: <enter>
```

13. If you have done everything correct, you will see:

```
Adding micex_vpn_connection - <server address> - test - PPTP
Added tunnel micex_vpn_connection
```

14. Type `‘8’` and press enter to exit the setup wizard.

15. The next step is to make a few changes in a configuration file which was created on previous steps by the wizard. At first open it using the following command:

```
sudo vim /etc/ppp/peers/micex_vpn_connection
```

16. Needed changes are colored by red:

```
#
# PPTP Tunnel configuration for tunnel micex_vpn_connection
# Server IP: <server address>
# Route: route del default
# Route: route add default gw 1.1.1.1 TUNNEL_DEV
#
noauth
```

```
#
# Tags for CHAP secret selection
#
name test
remotename PPTP

#
# Include the main PPTP configuration file
#
# file /etc/ppp/options.pptp
```

17. Please be careful and don’t forget to save this file before closing. That’s all. Now you are ready to establish the VPN connection using the following command:

    ```
    sudo /usr/sbin/pptp-command start micex_vpn_connection
    ```

    You will see something like this:

    ```
    Using interface ppp0
    Connect: ppp0 <--> /dev/pts/1
    local  IP address 1.1.1.19
    remote IP address 1.1.1.1
    Script ?? finished (pid 30023), status = 0x0
    Script /etc/ppp/ip-up finished (pid 30032), status = 0x0
    Route: add -net 0.0.0.0 gw 1.1.1.1 added
    Route: add -net 1.1.1.0 netmask 255.255.255.0 gw 1.1.1.1 added
    All routes added.
    Tunnel micex_vpn_connection is active on ppp0.  IP Address: 1.1.1.19
    ```

18. To stop this connection use the following command:

    ```
    sudo /usr/sbin/pptp-command stop
    ```

19. Important: After the VPN connection is stopped you will need to return the default route rule you had before. Otherwise the next tries to establish the VPN connection will be failed. It’s recommended to make a script which will be responsible for the default route rule restoring.

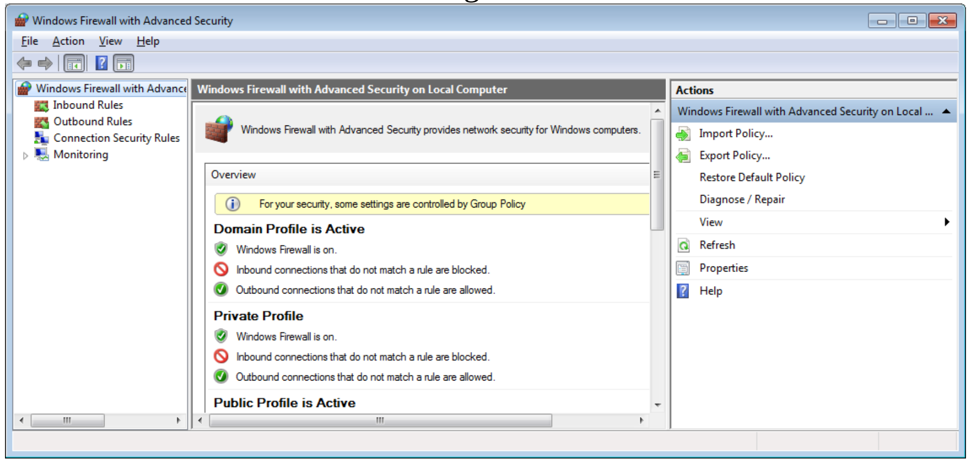
5.4.Troubleshooting

1. The VPN connection is established but your application doesn't receive UDP packets (Windows 7)

1.1 Open status of your VPN connection and check if the count of 'Received' bytes is continuously growing; If it's not so, ask for help the MOEX support team.

1.2 Check firewall settings. Temporary turn off the firewall. If after that all seems ok, turn on firewall again but add the firewall rule:

- ✓ Open *Windows Firewall→Advanced* settings;
- ✓ Choose *Inbound Rules* and on the right click *New Rule*:


Windows Firewall with Advanced Security
File
Action
View
Help
Windows Firewall with Advanced
Inbound Rules
Outbound Rules
Connection Security Rules
Monitoring
Windows Firewall with Advanced Security on Local Computer
Overview
Actions
Windows Firewall with Advanced Security on Local ...
Import Policy...
Export Policy...
Restore Default Policy
Diagnose / Repair
View
Refresh
Properties
Help

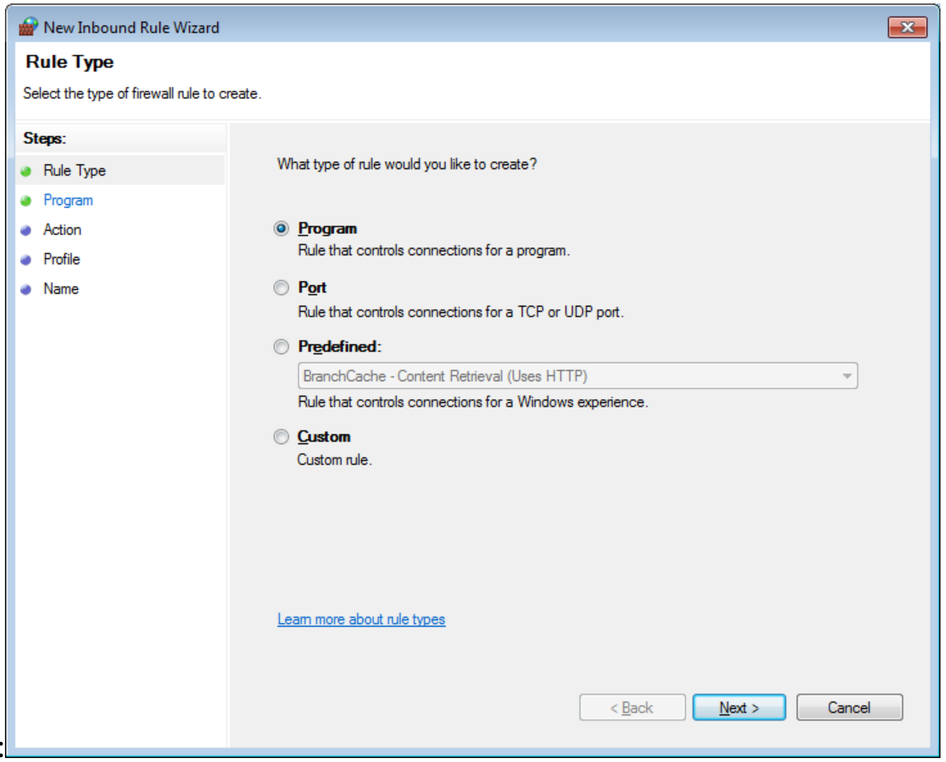
Windows Firewall with Advanced Security provides network security for Windows computers.

For your security, some settings are controlled by Group Policy

Domain Profile is Active
- Windows Firewall is on.
- Inbound connections that do not match a rule are blocked.
- Outbound connections that do not match a rule are allowed.

Private Profile
- Windows Firewall is on.
- Inbound connections that do not match a rule are blocked.
- Outbound connections that do not match a rule are allowed.

Public Profile is Active


New Inbound Rule Wizard

Rule Type

Select the type of firewall rule to create.

Steps:
- Rule Type
- Program
- Action
- Profile
- Name

What type of rule would you like to create?

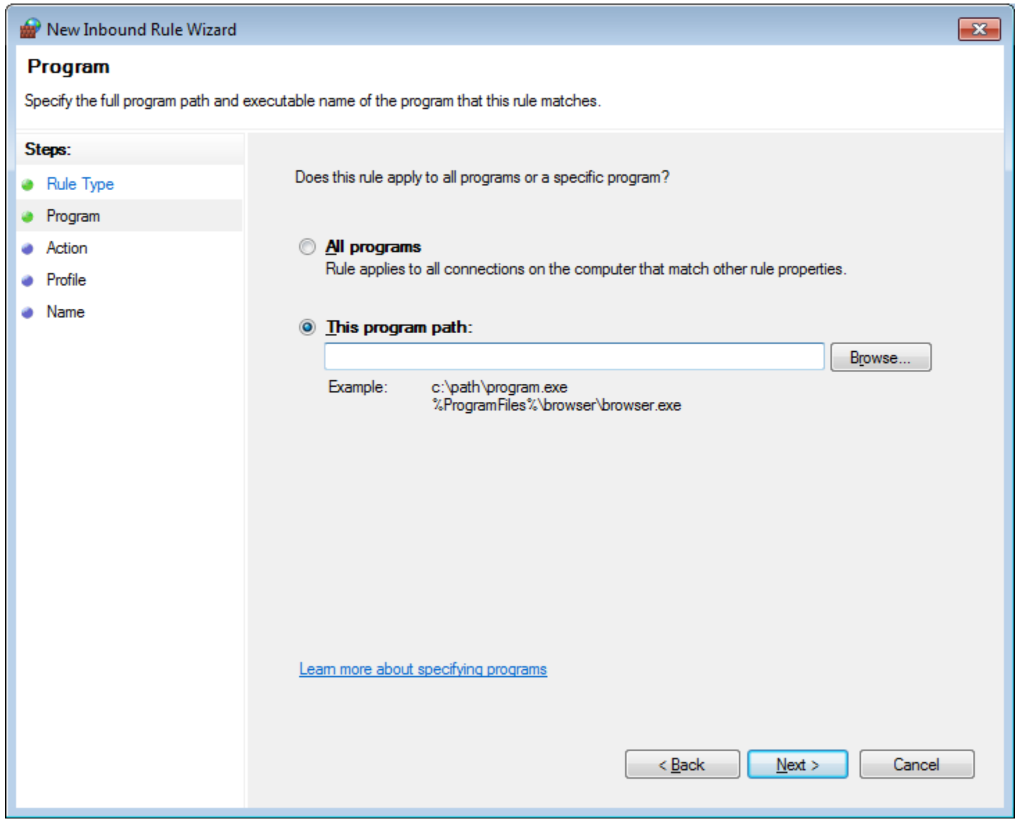
[✓ **Program**]
Rule that controls connections for a program.

Port
Rule that controls connections for a TCP or UDP port.

Predefined:
BranchCache - Content Retrieval (Uses HTTP)
Rule that controls connections for a Windows experience.

Custom
Custom rule.

- ✓ Leave the first page without changes and click *Next*:
- ✓ On the next page you need to specify path to your program:


New Inbound Rule Wizard

Program

Specify the full program path and executable name of the program that this rule matches.

Steps:
- Rule Type
- Program
- Action
- Profile
- Name

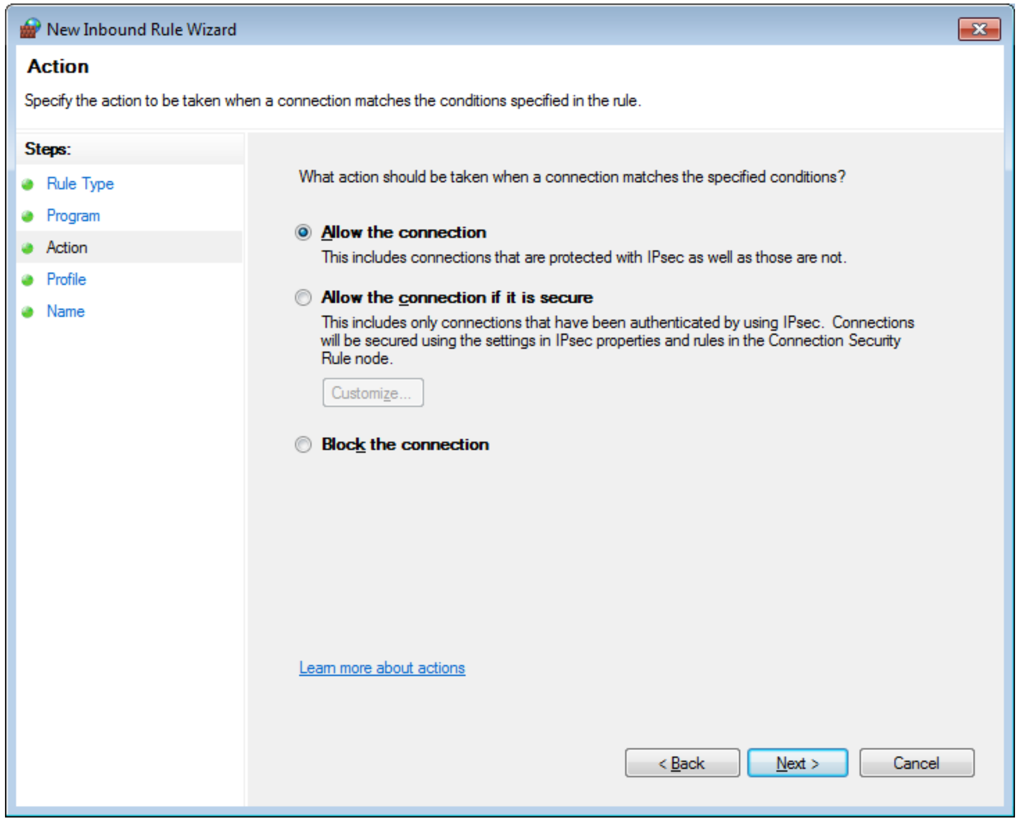
Does this rule apply to all programs or a specific program?

( ) **All programs**
Rule applies to all connections on the computer that match other rule properties.

(●) **This program path:**

Example: c:\path\program.exe
%ProgramFiles%\browser\browser.exe

✓ Leave the pages below without changes:


New Inbound Rule Wizard

Action

Specify the action to be taken when a connection matches the conditions specified in the rule.

Steps:

- Rule Type
- Program
- Action
- Profile
- Name

What action should be taken when a connection matches the specified conditions?

[✓ **Allow the connection**]
This includes connections that are protected with IPsec as well as those are not.

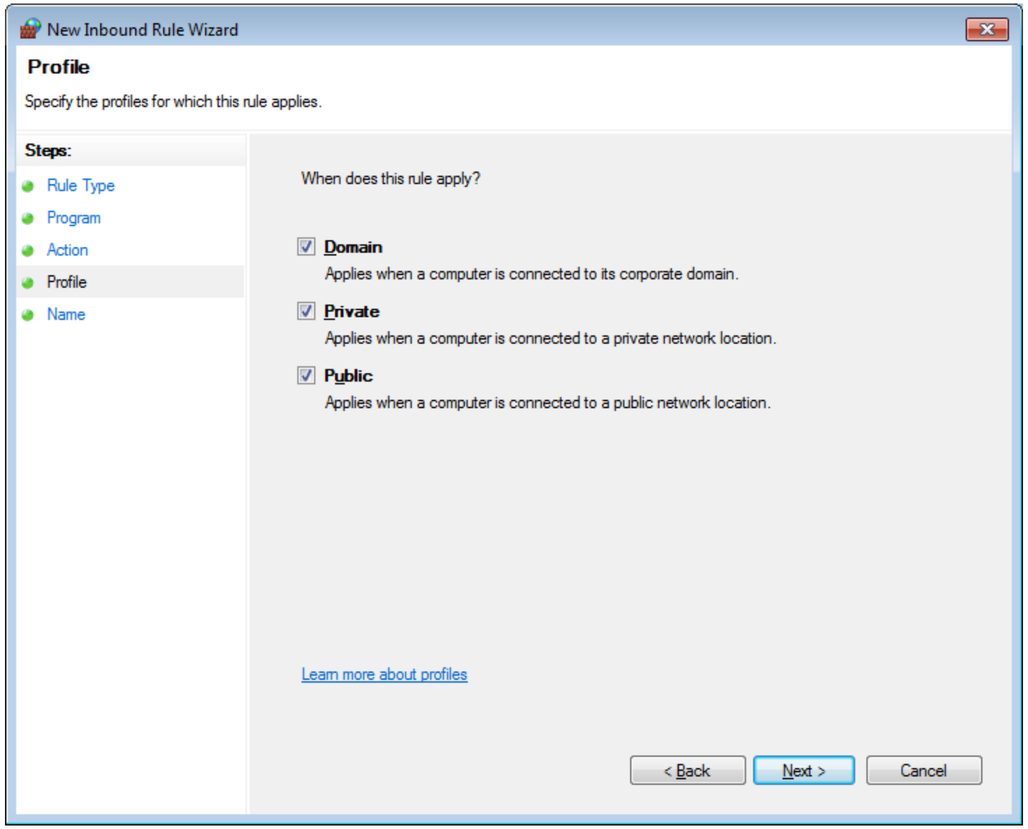
Allow the connection if it is secure
This includes only connections that have been authenticated by using IPsec. Connections will be secured using the settings in IPsec properties and rules in the Connection Security Rule node.

Customize...

Block the connection

Learn more about actions

< Back | Next > | Cancel



Profile

Specify the profiles for which this rule applies.

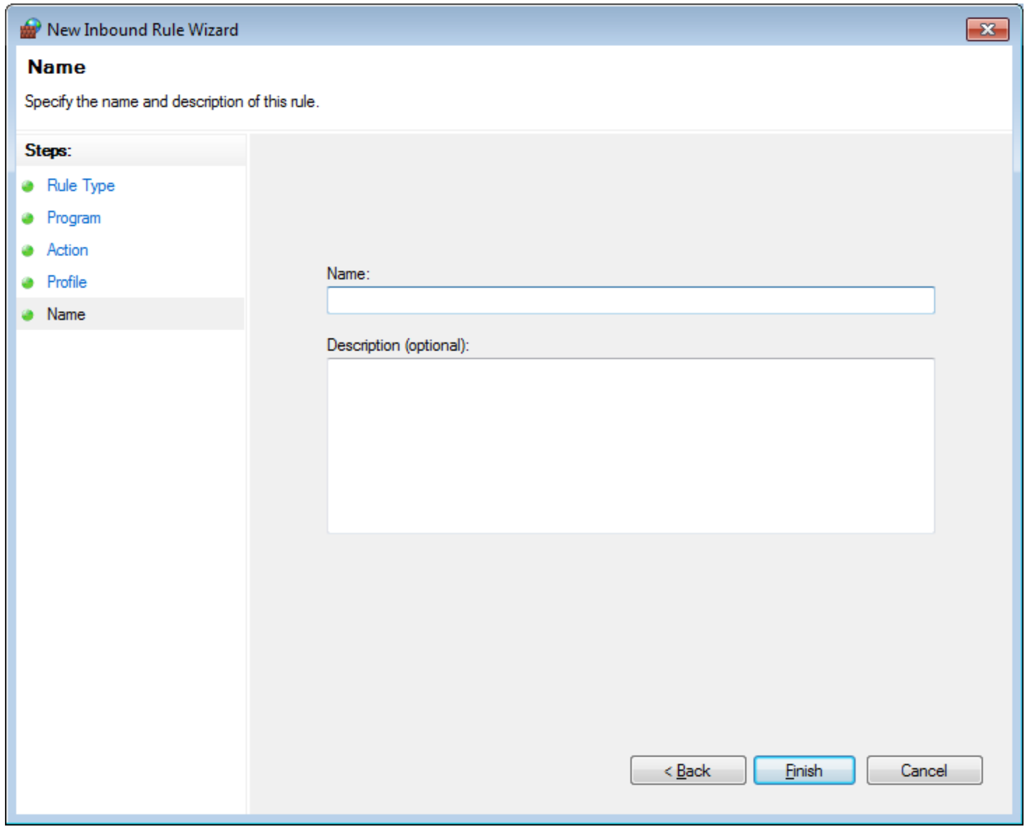
When does this rule apply?

[✓] **Domain**
Applies when a computer is connected to its corporate domain.

[✓] **Private**
Applies when a computer is connected to a private network location.

[✓] **Public**
Applies when a computer is connected to a public network location.

✓ Here you should to specify the name of this rule. E.g. MyApplicationRule.



Name

Specify the name and description of this rule.

Name:

Description (optional):

6. Certified Tools

6.1.EPAM – B2Bits ® FIX Antenna ™ Library

EPAM Systems, Inc., the leading software engineering and IT Outsourcing (ITO) provider in Central and Eastern Europe (CEE), and B2BITS®, EPAM Systems Capital Markets Competency Center, have certified their high performing FIX engine FIX Antenna ™ with MOEX Market Data Multicast FIX/FAST Platform.

FIX Antenna ^{TM} (C++ and .NET) allows one to transparently subscribe to Market Data from MOEX Market Data Multicast FIX/FAST Platform, hiding all feeds arbitraging and recovery functionality behind straightforward object oriented API. The package includes complete documentation and samples, illustrating the use of FIX Antenna ^{TM} with MOEX Market Data Multicast FIX/FAST Platform. .NET package also contains the GUI client, which receives Market Data from MOEX Market Data Multicast FIX/FAST Platform.

6.1.1 Quick Start – Code Samples

Here the source code of the simple client. This code is a skeleton of the real application and shows one of the possible ways to use FA micex_mfix.

instrument_listener_impl.h:

```
#pragma once

#include <B2BITS_micex_mfix_listeners.h>

namespace mfix_micex_client {
    class instrument_listener_impl
        : public micex_mfix::instrument_listener {
    public:
        virtual bool on_security_definition(const micex_mfix::security_description &sec_desc,
                                            const micex_mfix::security_id &sec_id,
                                            const micex_mfix::symbol &symb,
                                            const std::string &board,
                                            const Engine::FIXMessage &d_msg,
                                            const std::string &channel_id)
        {
            //add your processing code here, return your result true or false
            return false;
        }

        virtual void on_subscribed(const micex_mfix::symbol &symb,
                                   const std::string &board,
                                   micex_mfix::mfix_feed_type feed_type)
        {
            //add your processing code here
        }

        virtual void on_unsubscribed(const micex_mfix::symbol &symb,
                                     const std::string &board,
                                     micex_mfix::mfix_feed_type feed_type)
        {
            //add your processing code here
        }

        virtual void on_increment(const micex_mfix::symbol &symb,
                                  const std::string &board,
                                  const Engine::TagValue &entry,
                                  micex_mfix::mfix_feed_type feed_type)
        {
            //add your processing code here
        }

        virtual void on_security_status(const micex_mfix::symbol &symb,
                                        const std::string &board,
                                        const Engine::FIXMessage &msg,
                                        micex_mfix::mfix_feed_type feed_type)
        {
            //add your processing code here
        }

        virtual bool on_natural_refresh(const micex_mfix::symbol &symb,
                                        const std::string &board,
                                        const micex_mfix::increments &nr_msgs,
                                        micex_mfix::mfix_feed_type feed_type)
        {
            //add your processing code here, return your result true or false
            return true;
        }

        virtual void on_snapshot(const micex_mfix::symbol &symb,
                                 const std::string &board,
                                 const micex_mfix::snapshots &msgs,
                                 micex_mfix::mfix_feed_type feed_type)
        {
            //add your processing code here
        }

        virtual void on_recovery_started(const micex_mfix::symbol &symb,
                                         const std::string &board,
                                         micex_mfix::mfix_feed_type feed_type)
        {
            //add your processing code here, return your result true or false
            return false;
        }

        virtual void on_recovery_stopped(const micex_mfix::symbol &symb,
                                         const std::string &board,
                                         micex_mfix::mfix_recovery_reason reason,
                                         micex_mfix::mfix_feed_type feed_type)
        {
            //add your processing code here
        }

        virtual void on_error(const micex_mfix::symbol &symb,
                              const std::string &board,
                              const std::string &error,
                              micex_mfix::mfix_feed_type feed_type)
        {
            //add your processing code here
        }
    };
}
```

application_listener_impl.h:

```
#pragma once

#include <B2BITS_micex_mfix_listeners.h>

namespace mfix_micex_client {
    class application_listener_impl
        : public micex_mfix::micex_mfix_application_listener {
    public:
        virtual void on_error(const std::string &error)
        {
            //add your processing code here
        }

        virtual void on_process(const Engine::FIXMessage &msg, const std::string &channel_id)
        {
            //add your processing code here
        }

        virtual void on_feed_reset(const std::string &channel_id, micex_mfix::mfix_feed_type feed_type)
        {
            //add your processing code here
        }

        virtual void on_heartbeat(const std::string &channel_id, micex_mfix::mfix_feed_type feed_type)
        {
            //add your processing code here
        }
    };
}
```

main.cpp:

```
#include <iostream>

#include <B2BITS_FixEngine.h>
#include <B2BITS_micex_mfix_application.h>

#include "application_listener_impl.h"
#include "instrument_listener_impl.h"

using namespace mfix_micex_client;

void subscribe_and_wait(micex_mfix::micex_mfix_application *app,
```

```cpp
                        instrument_listener_impl *&ins_listener);

int main(int argc, char *agrv[])
{
    micex_mfix::micex_mfix_application *app = nullptr;
    application_listener_impl *app_listener = nullptr;
    instrument_listener_impl *ins_listener = nullptr;

    try {
        Engine::FixEngine::init("./engine.properties");

        //configure parameters
        micex_mfix::micex_mfix_application_params app_params;
        app_params.templates_fn_ = "./FIX50SP2.xml";
        app_params.config_xml_ = "./config.xml";

        app_listener = new application_listener_impl();
        app = Engine::FixEngine::singleton()->createMOEXApplication(app_params, app_listener);

        subscribe_and_wait(app, ins_listener);
    } catch (const Utils::Exception &ex) {
        std::cerr<<"Exception: "<<ex.what()<<"\n";
        if (nullptr != app_listener) {
            app_listener->release();
        }

        if (nullptr != ins_listener) {
            ins_listener->release();
        }

        return 100;
    }

    app_listener->release();
    ins_listener->release();

    app->release();

    return 0;
}

void subscribe_and_wait(micex_mfix::micex_mfix_application *app,
                        instrument_listener_impl *&ins_listener)
{
    //get channels id
    micex_mfix::channel_ids channels(app->get_channel_ids());

    //get orderbook feed
    micex_mfix::micex_feed &order_book_feed = app->get_orderbook_feed();
    ins_listener = new instrument_listener_impl();

    //subscribe to known instrument in channel[1], with market recovery as recovery type
    order_book_feed.subscribe_by_symbol("AFLT", "EQBR", *ins_listener,
                                    channels[1],micex_mfix::RM_USE_MARKET_RECOVERY);
    while (true) {
        std::cout<<"Type 'q' for exit\n\n";
        char c;
        std::cin>>c;
        if ('q' == c || 'Q' == c) {
            break;
        }
    }

    order_book_feed.unsubscribe_by_symbol("AFLT", "EQBR", channels[1]);
}
```

6.1.2 API Overview

Here is a list of all documented files with brief descriptions:

/include/B2BITS_micex_mfix_application.h
/include/B2BITS_micex_mfix_listeners.
/include/B2BITS_micex_mfix_types.h

Here are the classes, structs, unions and interfaces with brief descriptions:

micex_mfix::instrument_listener	instrument listener (observer)
micex_mfix::micex_feed	Represents micex feed (stream)
micex_mfix::micex_mfix_application	Represents micex mfix application
micex_mfix::micex_mfix_application_listener	Represents micex mfix application listener
micex_mfix::micex_mfix_application_params	Startup parameters
micex_mfix::security_definition_listener	Receives Security Definition messages

6.1.2.1.1. micex_mfix::instrument_listener Class Reference

```
#include <B2BITS_micex_mfix_listeners.h>
```

Public Member Functions

virtual void	**on_subscribed** (const symbol &symb, const std::string &board, mfix_feed_type feed_type)=0
	Faired when successfully subscribed to security description.
virtual void	**on_unsubscribed** (const symbol &symb, const std::string &board, mfix_feed_type feed_type)=0
	Faired when successfully unsubscribed from security description.
virtual void	**on_increment** (const symbol &symb, const std::string &board, const Engine::TagValue &entry, mfix_feed_type feed_type)=0
	Faired when user should reset book with the bnew values.
virtual void	**on_security_status** (const symbol &symb, const std::string &board, const Engine::FIXMessage &msg, mfix_feed_type feed_type)=0
	Faired when user should update instrument status.
virtual bool	**on_natural_refresh** (const symbol &symb, const std::string &board, const increments &nr_msgs, mfix_feed_type feed_type)=0 Faired when user should reset book with the bnew values and Natural Refresh is used return true if book is recovered otherwise false
virtual void	**on_snapshot** (const symbol &symb, const std::string &board, const snapshots &msgs, mfix_feed_type feed_type)=0
	Faired when user should reset book with the bnew values.
virtual void	**on_recovery_started** (const symbol &symb, const std::string &board, mfix_feed_type feed_type)=0
	Faired when recovery is started.
virtual void	**on_recovery_stopped** (const symbol &symb, const std::string &board,

	mfix_recovery_reason reason, mfix_feed_type feed_type)=0
	Faired when recovery is ended.
virtual void	**on_error** (const symbol &symb, const std::string &board, const std::string &error, mfix_feed_type feed_type)=0
	Faired on error (example: when second subscribing was attempt for the same instrument)

Note:
Objects of this class do not put to the std::auto_ptr or other smart pointers (except specialized, example Utils::RefCounterPtr). Object must be created via "new" keyword only.

6.1.2.1.2. micex_mfix::micex_feed Class Reference

```
#include <B2BITS_micex_mfix_application.h>
```

Public Member Functions

virtual void	**subscribe_by_symbol** (const symbol &symb, const std::string &board, instrument_listener &listener, const std::string &channel_id, mfix_recovery_mode recovery=RM_USE_MARKET_RECOVERY)=0
	Subscribes instrument by symbol.
virtual void	**unsubscribe_by_symbol** (const symbol &symb, const std::string &board, const std::string &channel_id)=0
	Unsubscribes from instrument by symbol.
virtual void	**subscribe_all** (instrument_listener &listener, const std::string &channel_id, mfix_recovery_mode recovery=RM_USE_MARKET_RECOVERY)=0
	Subscribe all instruments.
virtual void	**unsubscribe_all** (const std::string &channel_id)=0
	Unsubscribe all instruments.

6.1.2.1.3. micex_mfix::micex_mfix_application Class Reference

```
#include <B2BITS_micex_mfix_application.h>
```

Public Member Functions

virtual void	**release** ()=0
	Releases resources assigned to application.
virtual micex_feed &	**get_orderbook_feed** () const =0
	Retrieves order book feed (stream)
virtual micex_feed &	**get_statistics_feed** () const =0
	Retrieves statictics feed (stream)
virtual micex_feed &	**get_orders_feed** () const =0
	Retrieves order feed (stream)
virtual micex_feed &	**get_trades_feed** () const =0
	Retrieves trades feed (stream)
virtual const channel_ids &	**get_channel_ids** () const =0
	Returns channel ids.

6.1.2.1.4. micex_mfix::micex_mfix_application_listener Class Reference

```
#include <B2BITS_micex_mfix_listeners.h>
```

Public Member Functions

virtual void	**on_error** (const std::string &error)=0 Called on errors in micex mfix application This function can be called from different thread, so used should make it thread-safe in implementation
virtual void	**on_process** (const Engine::FIXMessage &msg, const std::string &channel_id)=0

	Called on non X, d and W messages This function can be called from different thread, so used should make it thread-safe in implementation
virtual void	**on_feed_reset** (const std::string &channel_id, mfix_feed_type feed_type)=0 Called on reset for feed (X-message was received with entry 269=J)
virtual void	**on_heartbeat** (const std::string &channel_id, mfix_feed_type feed_type)=0 Called on heartbeat messages

Note:

Objects of this class do not put to the std::auto_ptr or other smart pointers (except specialized, example Utils::RefCounterPtr). Object must be created via "new" keyword only

6.1.2.1.5. micex_mfix::micex_mfix_application_params Struct Reference

```
#include <B2BITS_micex_mfix_application.h>
```

Public Types

enum	**recovery_type** { **udp_recovery**, **tcp_recovery** }

Public Attributes

std::string	**templates_fn_** Path to the MFIX Market Data FAST templates file.
std::string	**config_xml_** Path to the MFIX Market Data configuration file.
size_t	**number_of_workers_** Number of threads to decode incoming data Default value is 4
size_t	**increment_queue_size_** Maximum number of messages could be stored in recovery mode for the particular instrument. Default value is 50
bool	**check_udp_sender_** Pass true to check the UDP packet sender's IP address. Default value is true

std::string	**listen_interface_ip_** IP of network interface to listen on; nullptr or empty string means all interfaces. Default value is null (all interfaces)
size_t	**incoming_udp_buffer_size_** UDP incoming buffer size. Should be tuned in case of UDP message miss
size_t	**application_message_queue_size_**
	Count of messages that are queued for processing by Application.
bool	**log_incoming_FIX_messages_** Pass true to write out to the log file incoming FIX messages Default value is false
bool	**log_incoming_udp_messages_** Pass true to write out to the binary log file incoming FAST messages Default value is false
std::size_t	**hole_pack_delay_** Number of incoming messages with seq num out of order to skip before start recovery. Default value is 1
recovery_type	**recovery_type_** Type for the recovery. tcp_recovery uses only tcp recovery for instruments (34 tag is used to detect hole) udp_recovery uses one mode of the mfix_recovery_mode for instruments (83 tag is used to detect hole) Default value is udp_recovery
std::string	**user_login_** User login for tcp recovery session Default value is empty string
std::string	**user_password_** User password for tcp recovery session Default value is empty string

6.1.2.1.6. micex_mfix::security_definition_listener Class Reference

```
#include <B2BITS_micex_mfix_listeners.h>
```

Public Member Functions

virtual bool	**on_security_definition** (const security_description &sec_desc, const security_id &sec_id, const symbol &symb, const std::string &board, const Engine::FIXMessage &d_msg, const std::string &channel_id)=0 Faired when security definition message was received Return true if need to continue listening instrument replay, false otherwise

Note:

Objects of this class do not put to the std::auto_ptr or other smart pointers (except specialized, example Utils::RefCounterPtr). Object must be created via "new" keyword only.