

MICEX Market Data Multicast FIX/FAST Platform

User Guide

Moscow Interbank Currency Exchange

Version 1.0

May 25, 2011

Contents

| | | |
|-------|--|----|
| 1. | Overview | 4 |
| 1.1. | Streaming Data | 4 |
| 1.2. | Incremental Messaging | 4 |
| 1.3. | FIX Format..... | 4 |
| 1.4. | FAST Compression | 4 |
| 1.5. | Multicast Delivery | 4 |
| 1.6. | Recovery | 5 |
| 2. | Getting Started with MICEX Market Data FIX/FAST Multicast Platform | 6 |
| 2.1. | Basic Scenario – Connect before the Trade Day Started | 6 |
| 2.2. | Connect after the Trade Day Started | 6 |
| 2.3. | Incremental Feeds A and B Arbitration | 6 |
| 3. | Core Functionality..... | 8 |
| 3.1. | Platform Architecture..... | 8 |
| 3.2. | FAST Implementation..... | 9 |
| 3.2.1 | Introduction | 9 |
| 3.2.2 | Stop Bit Encoding | 10 |
| 3.2.3 | Implicit Tagging..... | 10 |
| 3.2.4 | Field Encoding Operators | 10 |
| 3.2.5 | FAST Template | 10 |
| 3.2.6 | Decoding overview | 11 |
| 3.2.7 | Sample Template..... | 12 |
| 3.3. | Data Feeds..... | 15 |
| 3.3.1 | Instruments Feed | 15 |
| 3.3.2 | OrderBook, Market Statistics, Orders, and Trades Feeds | 15 |
| 3.3.3 | Market Recovery Feeds | 16 |
| 3.3.4 | TCP Replay | 16 |
| 3.4. | Recovery | 17 |
| 3.4.1 | Market Recovery Overview | 17 |
| 3.4.2 | Recovering Data – Process | 18 |
| 3.4.3 | TCP Replay | 19 |
| 4. | FIX Message Specification..... | 20 |
| 4.1. | FIX Component Blocks..... | 20 |
| 4.1.1 | Standard Message Header | 20 |
| 4.1.2 | Instrument | 20 |
| 4.1.3 | Instrument Leg | 22 |
| 4.1.4 | Instrument Extension | 22 |

| | | |
|-------|--|----|
| 4.1.5 | Underlying Instrument | 23 |
| 4.1.6 | Market Segment | 23 |
| 4.2. | FIX Session-Level Messages | 23 |
| 4.2.1 | Logon (A) | 23 |
| 4.2.2 | Logout (5) | 24 |
| 4.2.3 | Heartbeat (0) | 24 |
| 4.1. | FIX Application-Level Messages | 24 |
| 4.1.1 | Security Definition (d) | 24 |
| 4.1.2 | Security Status (f) | 25 |
| 4.1.3 | Trading Session Status (h) | 26 |
| 4.1.4 | Market Data Request (V) | 26 |
| 4.1.5 | Market Data - Snapshot/Full Refresh (W) | 26 |
| 4.1.6 | Market Data - Incremental Refresh (X) | 30 |
| 5. | Network Connectivity Guide | 33 |
| 5.1. | Configure a VPN connection with MICEX using Windows XP | 33 |
| 5.2. | Configure a VPN connection with MICEX using Windows 7 | 40 |
| 5.3. | Configure a VPN connection with MICEX using OpenSUSE | 45 |
| 5.4. | Troubleshooting | 47 |
| 6. | Certified Tools | 50 |
| 6.1. | EPAM – B2Bits ® FIX Antenna TM Library | 50 |
| 6.1.1 | Quick Start – Code Samples | 50 |
| 6.1.2 | API Overview | 53 |

1. Overview

This document describes the MICEX Market Data Multicast FIX/FAST Platform. This platform provides the new highly efficient mechanism for delivering MICEX 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.

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

1.1.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.

1.2.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.

1.3.FIX Format

MICEX 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 .

1.4.FAST Compression

FAST is a binary compression algorithm used to purpose 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 MFIX Market Data Multicast, see section 3.2. FAST Implementation.

1.5.Multicast Delivery

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

One FAST encoded FIX message cannot 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 MICEX 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 MICEX Market Data Multicast FIX/FAST Platform sends messages from different tables of the Trading System to different multicast groups.

1.6.Recovery

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

MICEX 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 MICEX Market Data FIX/FAST Multicast Platform

2.1. Basic Scenario – Connect before the Trade Day Started

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

The procedure is the following:

1. Download the actual 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 MICEX Market Data Multicast FIX/FAST Platform some time after the trading day started, it should keep the following procedure:

1. Download the actual 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. List feeds A and B
2. Process messages according their sequence numbers.
3. Ignore message if 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, because perhaps lost packet will come a bit late. UDP protocol can't guarantee sequence in delivery.

Example:

Feed A

Feed B

| |
|-------------------|
| 34-MsgSeqNum = 59 |
| 34-MsgSeqNum = 60 |
| 34-MsgSeqNum = 62 |
| 34-MsgSeqNum = 63 |
| 34-MsgSeqNum = 65 |

| |
|-------------------|
| 34-MsgSeqNum = 59 |
| 34-MsgSeqNum = 60 |
| 34-MsgSeqNum = 61 |
| 34-MsgSeqNum = 62 |
| 34-MsgSeqNum = 65 |

Messages are received from Feed A and Feed.

1. Receive message # 59 from Feed A, manage it.
2. Receive message #59 from Feed B, discard it, because this message were managed from Feed A.
3. Receive message # 60 from Feed A, manage it.
4. Receive message #60 from Feed B, discard it, because this message were managed from Feed A.
5. Receive message #62 from Feed A, discard it and wait for message #61.
6. Receive message # 61 from Feed B, manage it.
7. Receive message # 62 from Feed B, manage it.
8. Receive message #62 from Feed A, discard it, because this message were managed from Feed B.
9. Receive message # 63 from Feed A, manage 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.

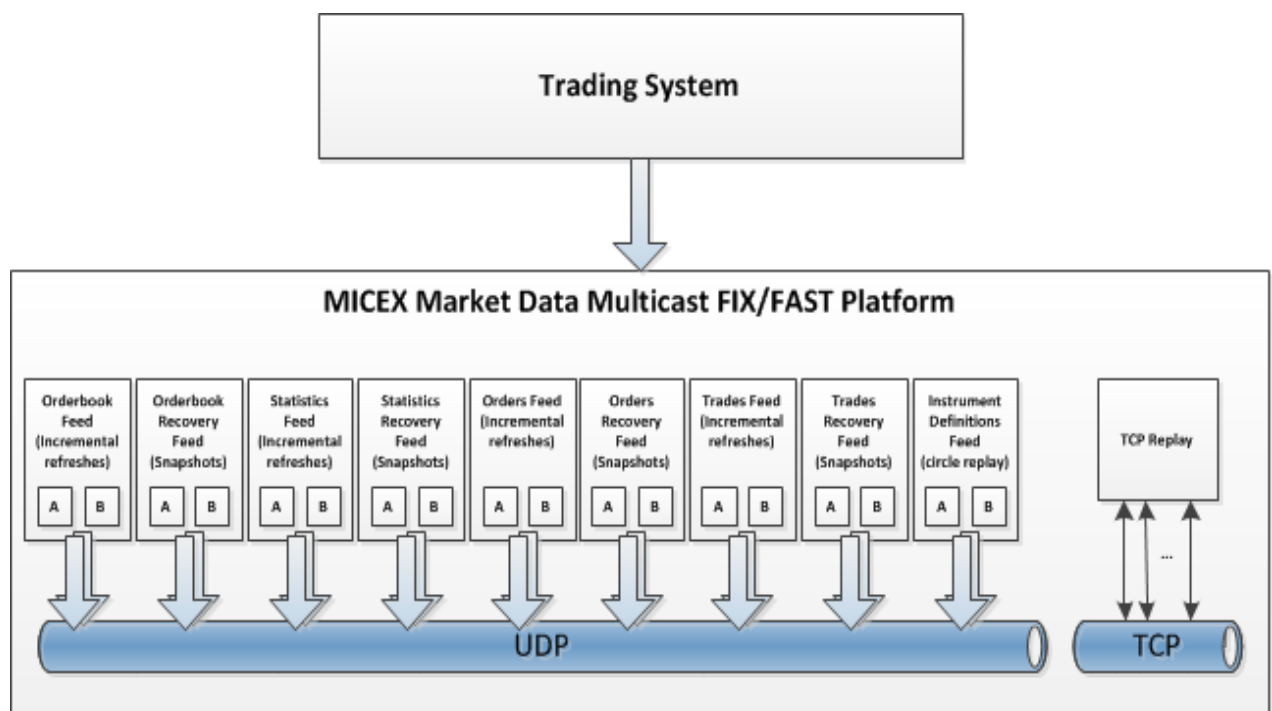
3. Core Functionality

3.1. Platform Architecture

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

Following feeds used in the system:

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



Error! Unknown document property name. broadcast feeds:

- Basic Feeds:
 - 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
- Recovery Feeds:

- OrderBook Recovery Feeds
 - OrderBook Recovery Feed A
 - OrderBook Recovery Feed B
- Statistics Recovery Feeds
 - Statistics Recovery Feed A
 - Statistics Recovery Feed B
- Orders Recovery Feeds
 - Orders Recovery Feed A
 - Orders Recovery Feed B
- Trades Recovery Feeds
 - Trades Recovery Feed A
 - Trades Recovery Feed B
- Instruments Definitions Feeds:
 - Instruments Definitions Feed A
 - Instruments Definitions Feed B

Besides market data transfer in UDP channels, MFIX Market Data Multicast can accept TCP requests from clients. Following feeds sends through TCP connection:

- OrderBook Feed
- Statistics Feed
- Orders Feed
- Trades Feed

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

1. available market data of the limited time interval;
2. fixed number of messages which is sending during one TCP session;
3. fixed number of messages requested through TCP connection during the trading day.

3.2.FAST Implementation

This part contains the description how to implement 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.

MFIX Market Data Multicast processed FIX messages which are encoded to FAST. The Preamble is found before the FAST encoded message, and contains the sequence number (Fig 1).

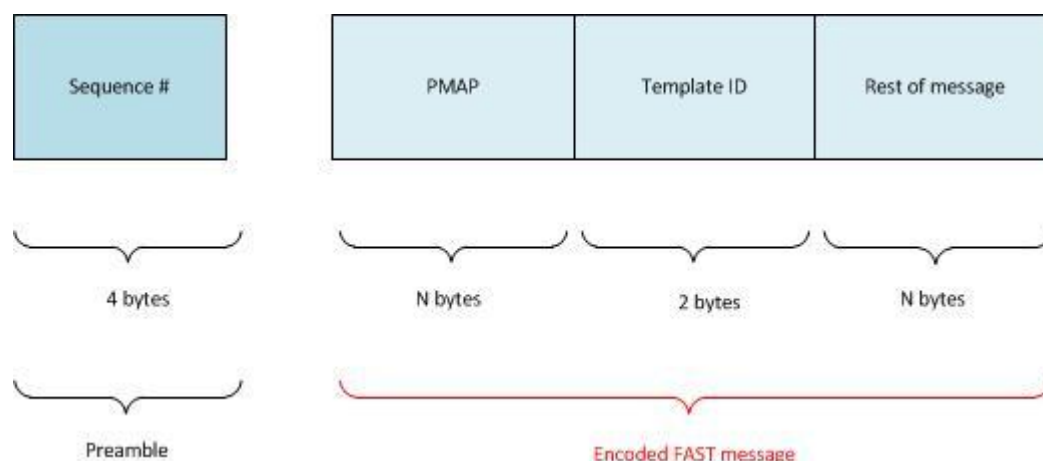


Figure 1

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, MICEX 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 for integers, Increment.

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

```
103 <!-- Market Data - Incremental Refresh -->
104 <template name="X" id="6" xmlns="http://www.fixprotocol.org/ns/fast/td/1.1">
105   <string name="MessageType" id="35">
106     <constant value="X"/>
107   </string>
108   <string name="ApplVerID" id="1128"><copy/></string>
109   <string name="SenderCompID" id="49"><copy/></string>
110   <uint32 name="MsgSeqNum" id="34"><increment/></uint32>
111   <uint64 name="SendingTime" id="52"><copy/></uint64>
112   <byteVector name="MessageEncoding" id="347" presence="optional"><default/></byteVector>
113   <sequence name="GroupMDEntries">
114     <length name="NoMDEntries" id="268"/>
115     <uint32 name="MDUpdateAction" id="279"><copy/></uint32>
116     <string name="MDEntryType" id="269" presence="optional"><copy/></string>
117     <byteVector name="MDEntryID" id="278" presence="optional"><copy/></byteVector>
118     <byteVector name="Symbol" id="55" presence="optional"><copy/></byteVector>
119     <int32 name="RptSeq" id="83" presence="optional"><copy/></int32>
120     <decimal name="MDEntryPx" id="270" presence="optional"><copy/></decimal>
121     <decimal name="MDEntrySize" id="271" presence="optional"><copy/></decimal>
122     <uint32 name="MDEntryDate" id="272" presence="optional"><copy/></uint32>
123     <uint32 name="MDEntryTime" id="273" presence="optional"><copy/></uint32>
124     <byteVector name="TradingSessionID" id="336" presence="optional"><copy/></byteVector>
125     <byteVector name="QuoteCondition" id="276" presence="optional"><copy/></byteVector>
126     <byteVector name="TradeCondition" id="277" presence="optional"><copy/></byteVector>
127     <uint32 name="OpenCloseSettleFlag" id="286" presence="optional"><default/></uint32>
128     <decimal name="NetChgPrevDay" id="451" presence="optional"><copy/></decimal>
129     <decimal name="Yield" id="236" presence="optional"><copy/></decimal>
130     <decimal name="AccruedInterestAmt" id="5384" presence="optional"><copy/></decimal>
131     <decimal name="ChgFromWAPrice" id="5510" presence="optional"><copy/></decimal>
132     <decimal name="ChgFromOpenInterest" id="5511" presence="optional"><copy/></decimal>
133     <int32 name="TotalNumOfTrades" id="6139" presence="optional"><copy/></int32>
134     <decimal name="TradeValue" id="6143" presence="optional"><copy/></decimal>
135     <int32 name="OfferNbOr" id="9168" presence="optional"><copy/></int32>
136     <int32 name="BidNbOr" id="9169" presence="optional"><copy/></int32>
137     <decimal name="ChgFromSettlmnt" id="9750" presence="optional"><copy/></decimal>
138     <int32 name="SumQtyOfBest" id="10503" presence="optional"><copy/></int32>
139     <string name="OrderSide" id="10504" presence="optional"><copy/></string>
140     <string name="OrdStatus" id="10505" presence="optional"><copy/></string>
141     <decimal name="OrdBalance" id="10506" presence="optional"><copy/></decimal>
142     <decimal name="OrdValue" id="10507" presence="optional"><copy/></decimal>
143     <decimal name="MinCurrPx" id="10509" presence="optional"><copy/></decimal>
144     <uint32 name="MinCurrPxChgTime" id="10510" presence="optional"><copy/></uint32>
145   </sequence>
146 </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:

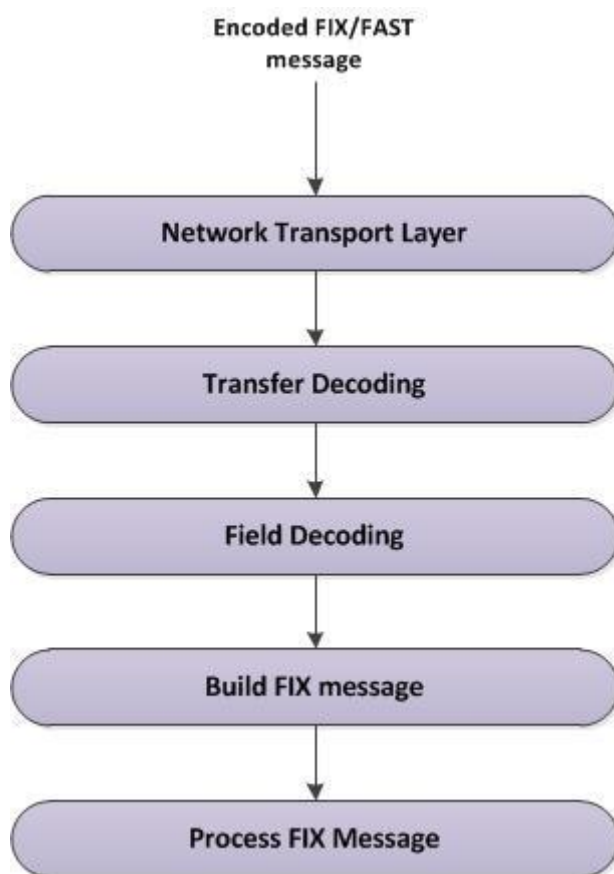


Figure 3

- **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's include:
 - 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 is include:
 - 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 deserialized 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

| Line # | 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" id="347" presence="optional"><default/></byteVector> | Field instruction for MessageEncoding defined as a byte vector and with identifier = 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 | <decimal name="ChgOpenInterest" id="5511" presence="optional"><copy/></decimal> | Field instruction for ChgOpenInterest defined as a decimal with identifier = 5511 corresponding to the FIX custom tag number. ChgOpenInterest has a copy field operator. |
| 26 | <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. |
| 27 | <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. |
| 28 | <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. |
| 29 | <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. |
| 30 | <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. |
| 31 | <decimal name="ChgFromSettlmnt" id="9750" presence="optional"><copy/></decimal> | Field instruction for ChgFromSettlmnt defined as a decimal with identifier = 9750 corresponding to the FIX custom tag number. ChgFromSettlmnt has a copy field operator. |
| 32 | <int32 name="SumQtyOfBest" id="10503" presence="optional"><copy/></int32> | Field instruction for SumQtyOfBest defined as a signed integer with identifier = 10503. SumQtyOfBest has a copy field operator. |
| 33 | <string name="OrderSide" id="10504"> | Field instruction for OrderSide defined as a string with |

| | | |
|----|---|---|
| | presence="optional"><copy/></string> | an identifier = 10504. OrderSide has a copy field operator. |
| 34 | <string name="OrdStatus" id="10505" presence="optional"><copy/></string> | Field instruction for OrdStatus defined as a string with an identifier = 10505. OrdStatus has a copy field operator. |
| 37 | <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. |
| 38 | <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 a highly optimized feeds which is distributed in UDP channels. Each Feed transferred into 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 sends equal market data information. It's provides low probability of packet's lost, and reduce needs 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 sends 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=MICEX|55=VRSB
P|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"
OAO                                     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=FOND|SMAL|10=
000|
```

3.3.2 OrderBook, Market Statistics, Orders, and Trades Feeds

Following market data also distributed in separate feeds:

- OrderBook Feed A/B – changes of 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).

- 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' (Trade), '3' (Index value), '4' (Opening price), '5' (Closing price), '6' (Settlement price), '7' (Trading session high price), '8' (Trading session low price), '9' (Trading session VWAP price), 'B' (Trade volume), 'C' (Open interest), 'N' (Session high bid), 'O' (Session low offer), 'i' (Last bid price), 'j' (Last offer price), 'h' (Open period price), 'k' (Close period price), 'l' (Market price 2), 'm' (Market price), 'n' (Last negotiated deal), 'o' (Official open price), 'p' (Official current price), 'r' (Official close price), 'v' (Total bid volume), 'w' (Total offer volume), 't' (Negotiated deals value), 'u' (Duration).
- Orders Feed A/B – changes of ORDERS table.
Orders Feeds also include Add, Change, and Delete blocks. Entry types are: '0' (Bid), '1' (Offer).
- Trades Feed A/B – changes of 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 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 MICEX market. When status of connection changed this message is sending into UDP channel. When status of a security changed Security Status (f) message is sending 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) message encoded to FAST. One message contain information about one security. Information in Market Data Snapshot Full Refresh message include connection status 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 client systems have retrieved recovery data, client systems should stop listening to the Market Recovery feeds.

3.3.4 TCP Replay

The TCP replay component allows you to request a replay of a set of messages already published on 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. Client can request the fixed number of messages.

Request is sent through a new TCP connection established by the customer. The responses are sent by MICEX through this same connection and the connection is then closed by MICEX once the replay is complete.

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

3.4.Recovery

MICEX 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 provide 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.

MICEX Market Data Multicast 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

Recovery method preferably 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 queue real-time data until all missed data is recovered. The recovered data should then be applied prior to queued data.

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 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.

3.4.3 TCP Replay

Market data from OrderBook, Statistics, Orders, Trades Feeds that were missed can be recover using the sequence number and the TCP historical replay component. 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 specified in the MICEX Market Data Multicast configuration file.

TCP replay include follows:

1. Establish TCP connection with MICEX Market Data Multicast.
2. Send FIX message Logon(A) to server. After success authorization server send FAST encoded Logon(A) message.
3. Send Market Data Request (V) message with:
 - a. Tag ApplID (1180) - the channel ID.
 - b. Range of sequence numbers - ApplBegSeqNum(1182) and ApplEndSeqNum (1183) tags.

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

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

After server response connection is closed.

Server will process only first user request, second and others will be ignored.

3.4.4

4. FIX Message Specification

This part contains the description of FIX 5.0 SP2 protocol messages, component blocks and fields which are supported by MFIX 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 MICEX'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 MFIX Market Data Multicast. Note that all fields which are required or conditionally required by FIX 5.0 SP2 standard but absent in MICEX Interface specification are optional and **will be ignored by MICEX**. 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 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 MICEX, then it should contain USERID assigned to a trader by MICEX. |
| 34 | MsgSeqNum | Y | SeqNum | | Integer message sequence number. Can be embedded within encrypted data section. |
| 52 | SendingTime | Y | UTCTimestamp | | Time of message transmission (expressed in UTC). YYYYMMDD-HH:MM:SS.sss Can be embedded within encrypted data section. |
| 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 MICEX internal instrument identifier, SecCode. |
| 48 | SecurityID | N | String | | Security identifier value of SecurityIDSource (22) type. |
| 22 | SecurityIDS | N | String | '4' (ISIN) | Identifies class or source of the SecurityID (48) |

| | source | | | | value. |
|-----|-------------------|---|---------------|--|--|
| 460 | Product | N | int | '3' (CORPORATE); '4' (CURRENCY); '5' (EQUITY); '6' (GOVERNMENT); '7' (INDEX); '11' (MUNICIPAL); '13' (FINANCING). | Indicates the type of product the security is associated with. |
| 461 | CFI Code | 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); 'BN' (Bank Notes); 'MF' (Mutual Fund); 'MLEG' (Multileg Instrument); 'MUNI' (Municipal bonds). | Indicates type of security. |
| 200 | MaturityMonthYear | N | month-year | | This field provides the actual calendar date for contract maturity – month and year (used for standardized futures and options) Format: YYYYMM. |
| 541 | MaturityDate | N | LocalMkt Date | | Date of maturity. |
| 224 | CouponPaymentDate | N | LocalMkt Date | | Date interest is to be paid. |
| 228 | Factor | N | float | | For Fixed Income: Amorization Factor for deriving Current face from Original face for ABS or MBS securities, note the fraction may be greater than, equal to or less than 1. In TIPS securities this is the Inflation index. Qty * Factor * Price = Gross Trade Amount For Derivatives: Contract Value Factor by which price must be adjusted to determine the true nominal value of one futures/options contract. (Qty * Price) * Factor = Nominal Value |
| 202 | StrikePrice | N | Price | | Strike Price for an Option. |
| 223 | CouponRate | N | Percentage | | The rate of interest. |

| | | | | | |
|-----------|-----------------------------|---|--------------|---|--|
| 107 | SecurityDesc | N | String | | Security description. |
| 350 | EncodedSecurityDescLen | N | Length | | Byte length of encoded (non-ASCII characters) EncodedSecurityDesc (351) field. |
| 351 | EncodedSecurityDesc | N | data | | Encoded (non-ASCII characters) representation of the SecurityDesc (107) field in the encoded format specified via the MessageEncoding (347) field. If used, the ASCII (English) representation should also be specified in the SecurityDesc (107) field. |
| 864 | NoEvents | N | NumInGroup | | Number of repeating EventType (865) entries. |
| => 865 | EventType | N | int | '7' (Last Eligible Trade Date); '100' (First Eligible Trade Date). | Code to represent the type of event Required if NoEvents (864) > 0. |
| => 866 | EventDate | N | LocalMktDate | | Date of event. |
| 521 7 | StateSecurityID | N | String | | State Securities Identification Number. |
| 538 2 | EncodedShortSecurityDescLen | N | Length | | Byte length of encoded (non-ASCII characters) EncodedShortSecurityDesc (5383) field. |
| 538 3 | EncodedShortSecurityDesc | N | data | | Encoded (non-ASCII characters) representation of the ShortSecurityDesc (5381) field in the encoded format specified via the MessageEncoding (347) field. |
| 555 6 | BaseSwapPx | N | Price | | Base SWAP price. |
| 555 8 | BuyBackPx | N | Price | | Buy back price. |
| 555 9 | BuyBackDate | N | LocalMktDate | | Buy back date. |
| 567 7 | Repo2Px | N | Price | | Price of the second part of REPO. |

4.1.3 Instrument Leg

Table 4

| Tag | Field name | Req'd | Type | Valid values | Comments |
|-----|-----------------|-------|----------|-----------------------------------|--|
| 600 | LegSymbol | Y | String | | Multileg instrument's individual securitys Symbol. |
| 607 | LegProduct | N | int | '4' (CURRENCY) | Multileg instrument's individual securitys Product. |
| 608 | LegCFICode | N | String | | Multileg instrument's individual securitys CFICode. |
| 609 | LegSecurityType | N | String | 'FOR' (Foreign Exchange Contract) | Multileg instrument's individual securitys SecurityType. |
| 556 | LegCurrency | N | Currency | | Currency associated with a particular Leg's quantity |
| 587 | LegSettlType | N | char | '1' (Cash); '2' (Next day). | Indicates order settlement period for Multileg instrument. |

4.1.4 Instrument Extension

Table 5

| 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. |

4.1.5 Underlying Instrument

Table 6

| Tag | Field name | Req'd | Type | Valid values | Comments |
|-----|-------------------|-------|--------|--------------|------------------------------|
| 311 | Underlying Symbol | Y | String | | Underlying securitys Symbol. |

4.1.6 Market Segment

Table 7

| 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 | | Identifier for Trading Session is used to represent SECBOARD. |

4.2.FIX Session-Level Messages

4.2.1 Logon (A)

Logon message from customer to MICEX:

Table 8

| 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 MICEX to customer:

Table 9

| 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 10

| 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 11

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

4.1.FIX Application-Level Messages

4.1.1 Security Definition (d)

Table 12

| Tag | Field name | Req'd | Type | Valid values | Comments |
|--|---------------|-------|------------|--------------|---|
| <Standard Message Header> | | Y | | | MsgType = 'd' |
| 911 | TotNumReports | Y | int | | Total number of reports returned in response to a request. |
| 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. |
| 711 | NoUnderlyings | N | NumInGroup | | Number of underlying legs that make up the security. |
| => component block <Underlying Instrument> | | N | | | The <UnderlyingInstrument> component block contains all the fields commonly used to describe a security or instrument. Required if NoUnderlyings (711) > 0. |
| 555 | NoLegs | N | NumInGroup | | Number of InstrumentLeg repeating group instances. |

| | | | | | |
|--|---------------------|---|------------|--------------------------------------|--|
| | | | up | | |
| => component block <Instrument Leg> | | N | | | <InstrumentLeg> component block describes a security used in multileg-oriented messages. Required if NoLegs (555) > 0. |
| 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. |
| 120 | SettlCurrency | N | Currency | | Currency code of settlement denomination. |
| 423 | PriceType | N | int | '1' (Percentage); '2' (Per unit). | Code to represent the price type. |
| 538 5 | MarketCode | N | String | | Code of market where instrument is traded. |
| 969 | MinPriceIncrement | N | float | | Minimum price increase for a given exchange-traded Instrument. |
| 538 7 | MktShareLimit | N | Percentage | | Market share limit. |
| 538 8 | MktShareThreshold | N | Qty | | Market share limit threshold. |
| 538 9 | MaxOrdersVolume | N | Qty | | Market share limit threshold. |
| 547 0 | PriceMvmLimit | N | Price | | Maximum deviation of prices from settlement price. |
| 550 8 | FaceValue | N | Amt | | Maximum deviation of prices from settlement price. |
| 759 5 | NoSharesIssued | N | Qty | | The number of shares issued. |
| 919 9 | HighLimit | N | Price | | Maximum authorized price at which an instrument can trade. |
| 920 0 | LowLimit | N | Price | | Minimum authorized price at which an instrument can trade. |
| 105 08 | NumOfDaysToMaturity | N | int | | The number of days to maturity of the security. |

4.1.2 Security Status (f)

Table 13

| 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 MICEX internal instrument identifier, SecCode. |
| 336 | TradingSessionID | N | String | | Identifier for Trading Session is used to represent SECBOARD. |
| 326 | SecurityTradingStatus | N | int | '2' (Trading halt); '17' (Ready to trade (start of session)); '18' (Not available for trading (end of session)); '20' (Unknown or Invalid). | Identifies the trading status applicable to the transaction. |
| 550 | AuctionInd | N | Boolean | 'Y' (Yes); | Indicates whether or not the auction is being held for |

| | | | | | |
|---|--------|--|--|-----------|---------------|
| 9 | icator | | | 'N' (No). | the security. |
|---|--------|--|--|-----------|---------------|

4.1.3 Trading Session Status (h)

Table 14

| 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 | TradSessionStatus | Y | int | '100' (Connection to MICEX market established); '101' (Lost connection to MICEX); '102' (Connection to MICEX market established, trading system wasn't restarted); '103' (Connection to MICEX market established, trading system was restarted). | State of the trading session. |
| 58 | Text | N | String | | Free format text string. |

4.1.4 Market Data Request (V)

Table 15

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

4.1.5 Market Data - Snapshot/Full Refresh (W)

Table 16

| 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. |

| | | | | | |
|--------|-------------------------|---|-------------|---|---|
| 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 MICEX market established); '101' (Lost connection to MICEX); '102' (Connection to MICEX market established, trading system wasn't restarted); '103' (Connection to MICEX market established, trading system was restarted). | State of the trading session. |
| 55 | Symbol | Y | String | | Ticker symbol. The MICEX internal instrument identifier, SecCode. |
| 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. |
| 1682 | MDSecurityTradingStatus | N | int | '2' (Trading halt); '17' (Ready to trade (start of session)); '18' (Not available for trading (end of session)); '20' (Unknown or Invalid). | Identifies the trading status applicable to the transaction. |
| 5509 | AuctionIndicator | N | Boolean | 'Y' (Yes); 'N' (No). | Indicates whether or not the auction is being held for the security. |
| 451 | NetChgPrevDay | N | PriceOffset | | Net change from previous days 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' (Trade); '3' (Index Value); '4' (Opening Price); '5' (Closing Price); '6' (Settlement Price); '7' (Trading Session High Price); '8' (Trading Session Low Price); '9' (Trading Session VWAP | Type Market Data entry. |

| | | | | | |
|--------|------------------|---|---------------------|---|--|
| | | | | Price); 'B' (Trade Volume); 'C' (Open Interest); ' <i>J</i> ' (<i>Empty book</i>); 'N' (Session high bid); 'O' (Session low offer); 'i' (Last bid price); 'j' (Last offer price); 'h' (Open period price); 'k' (Close period price); 'l' (Market price 2); 'm' (Market price); 'n' (Last negotiated deal); 'o' (Official open price); 'p' (Official current price); 'r' (Official close price); 'v' (Total bid volume); 'w' (Total offer volume); 't' (Negotiated deals value); 'u' (Duration); 'z' (Trade list). | |
| => 278 | MDEntryID | N | String | | Unique Market Data Entry identifier. Used, for example, for TRADENO. |
| => 270 | MDEntryPx | C | Price | | Price of the Market Data Entry. Required if MDEntryType (269) not in ('A', 'B', 'C', 'J'). |
| => 271 | MDEntrySize | C | Qty | | Orderbook quantity represented by the Market Data Entry. Required if MDEntryType (269) in ('0', '1', '2', 'B', 'C'). |
| => 272 | MDEntryDate | N | UTCDate Only | | Date of Market Data Entry. |
| => 273 | MDEntryTime | N | UTCTime Only | | Time of Market Data Entry. |
| => 336 | TradingSessionID | N | String | | Identifier for Trading Session is used to represent SECBOARD. |
| => 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)); | Space-delimited list of conditions describing a trade. |

| | | | | | |
|-----------------|------------------------|---|-------------------------|---|--|
| | | | | 'R' (Opening Price) ; 'AJ' (Official Closing Price); '98' (Minimum index value); '99' (Maximum index value). | |
| => 286 | OpenClose SettlFlag | N | MultipleV alueString | '4' (Entry from previous business day) | Flag that identifies a market data entry. |
| => 236 | Yield | N | Percentage | | Yield percentage. |
| => 538 4 | AccruedInt erestAmt | N | Amt | | Amount of accrued interest. |
| => 551 0 | ChgFrom WAPrice | N | PriceOffset | | Indicates change from previous day's weighted average price vs. last traded price. |
| => 551 1 | ChgOpenI nterest | N | Qty | | Indicates change from previous day's open interest. |
| => 613 9 | TotalNum OfTrades | N | int | | Total number of trades. |
| => 614 3 | TradeValu e | N | Amt | | Trade Value. |
| => 916 8 | OfferNbOr | N | int | | Number of sell orders. |
| => 916 9 | BidNbOr | N | int | | Number of buy orders. |
| => 975 0 | ChgFromS ettlmnt | N | PriceOffset | | Indicates the change in an instrument price from the previous day's settlement price |
| => 105 03 | SumQtyOf Best | N | int | | Cumulative sum of the best quotes. |
| => 105 04 | OrderSide | N | char | | Side of order. |
| => 105 05 | OrdStatus | N | char | 'O' (Active); 'M' (Matched); 'W' (Withdrawn); 'F' (Rejected by counterparty); 'R' (Rejected by the Trading System); 'C' (Cancelled by the Trading System); 'T' (Order activation time hasn't come yet). | Describes the current state of order. |
| => 105 09 | MinCurrPx | N | Price | | Minimum current price. |
| => | MinCurrPx | N | UTCTime | | Time when minimum current price was changed. |

| | | | | | |
|-----------|---------|--|------|--|--|
| 105 10 | ChgTime | | Only | | |
|-----------|---------|--|------|--|--|

4.1.6 Market Data - Incremental Refresh (X)

Table 17

| 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' (Trade); '3' (Index Value); '4' (Opening Price); '5' (Closing Price); '6' (Settlement Price); '7' (Trading Session High Price); '8' (Trading Session Low Price); '9' (Trading Session VWAP Price); 'B' (Trade Volume); 'C' (Open Interest); 'N' (Session High Bid); 'O' (Session Low Offer); 'i' (<i>Last bid price</i>); 'j' (<i>Last offer price</i>); 'h' (<i>Open period price</i>); 'k' (<i>Close period price</i>); 'l' (<i>Market price 2</i>); 'm' (<i>Market price</i>); 'n' (<i>Last negotiated deal</i>); 'o' (<i>Official open price</i>); 'p' (<i>Official current price</i>); | Type Market Data entry. |

| | | | | | |
|---------|--------------------------------|---|---------------------|---|--|
| | | | | <i>'r' (Official close price);</i> <i>'v' (Total bid volume);</i> <i>'w' (Total offer volume);</i> <i>'t' (Negotiated deals value);</i> <i>'u' (Duration);</i> <i>'z' (Trade list).</i> | |
| => 278 | MDEntryID | N | String | | Unique Market Data Entry identifier. Used, for example, for TRADENO. |
| => 55 | Symbol | Y | String | | Ticker symbol. The MICEX internal instrument identifier, SecCode. |
| => 83 | RptSeq | Y | int | | Sequence number of message within report series. |
| => 270 | MDEntryPx | C | Price | | Price of the Market Data Entry. Required if MDEntryType (269) not in ('A', 'B', 'C', 'J'). |
| => 271 | MDEntrySize | C | Qty | | Orderbook quantity represented by the Market Data Entry. Required if MDEntryType (269) in ('0', '1', '2', 'B', 'C'). |
| => 272 | MDEntryDate | N | UTCDate Only | | Date of Market Data Entry. |
| => 273 | MDEntryTime | N | UTCTime Only | | Time of Market Data Entry. |
| => 336 | TradingSessionID | N | String | | Identifier for Trading Session is used to represent SECBOARD. |
| => 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 index value); '99' (Maximum index value). | Space-delimited list of conditions describing a trade. |
| => 286 | OpenCloseSettleFlag | N | MultipleValueString | '4' (Entry from previous business day) | Flag that identifies a market data entry. |
| => 451 | NetChangePreviousDay | N | PriceOffset | | Net change from previous days closing price vs. last traded price. |
| => 236 | Yield | N | Percentage | | Yield percentage. |
| => 5384 | AccruedInterestAmt | N | Amt | | Amount of accrued interest. |
| => 5510 | ChangeFromWeightedAveragePrice | N | PriceOffset | | Indicates change from previous day's weighted average price vs. last traded price. |
| => | ChangeOpenInterest | N | Qty | | Indicates change from previous day's open interest. |

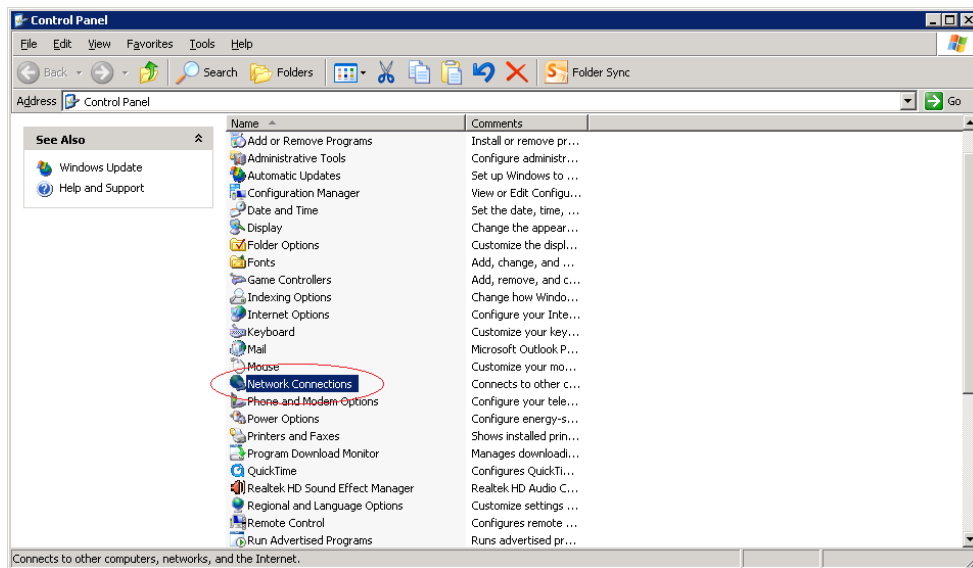
| | | | | | |
|---------|---------------------|---|--------------|---|--|
| 5511 | nInterest | | | | |
| =>6139 | TotalNumOfTrades | N | int | | Total number of trades. |
| =>6143 | TradeValue | N | Amt | | Trade Value. |
| =>9168 | OfferNbOr | N | int | | Number of sell orders. |
| =>9169 | BidNbOr | N | int | | Number of buy orders. |
| =>9750 | ChgFromSettlement | N | PriceOffset | | Indicates the change in an instrument price from the previous day's settlement price |
| =>10503 | SumQtyOfBest | N | int | | Cumulative sum of the best quotes. |
| =>10504 | OrderSide | N | char | | Side of order. |
| =>10505 | OrdStatus | N | char | 'O' (Active); 'M' (Matched); 'W' (Withdrawn); 'F' (Rejected by counterparty); 'R' (Rejected by the Trading System); 'C' (Cancelled by the Trading System); 'T' (Order activation time hasn't come yet). | Describes the current state of order. |
| =>10509 | MinCurrentPx | N | Price | | Minimum current price. |
| =>10510 | MinCurrentPxChgTime | N | UTCTime Only | | Time when minimum current price was changed. |

5. Network Connectivity Guide

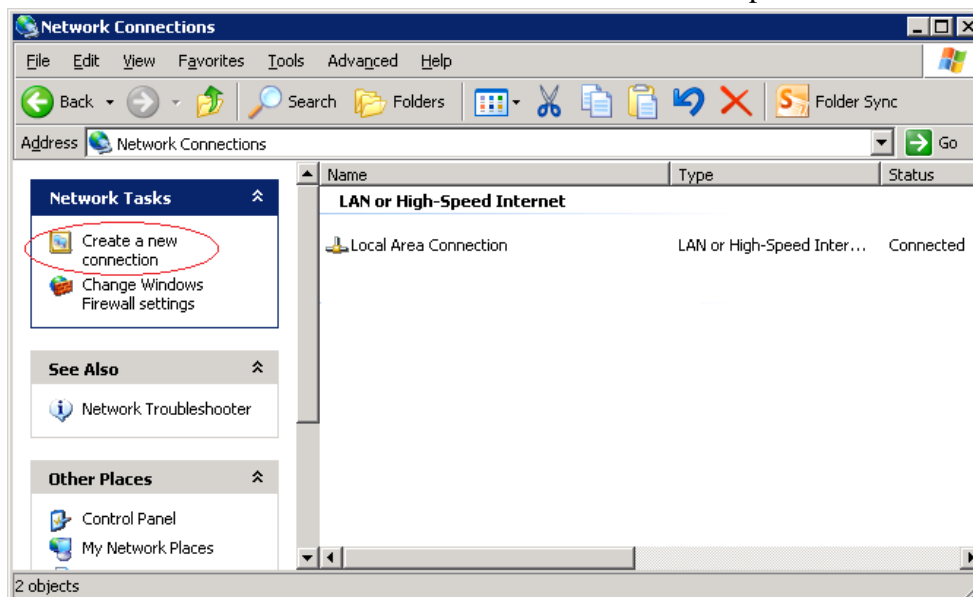
5.1. Configure a VPN connection with MICEX using Windows XP

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*:



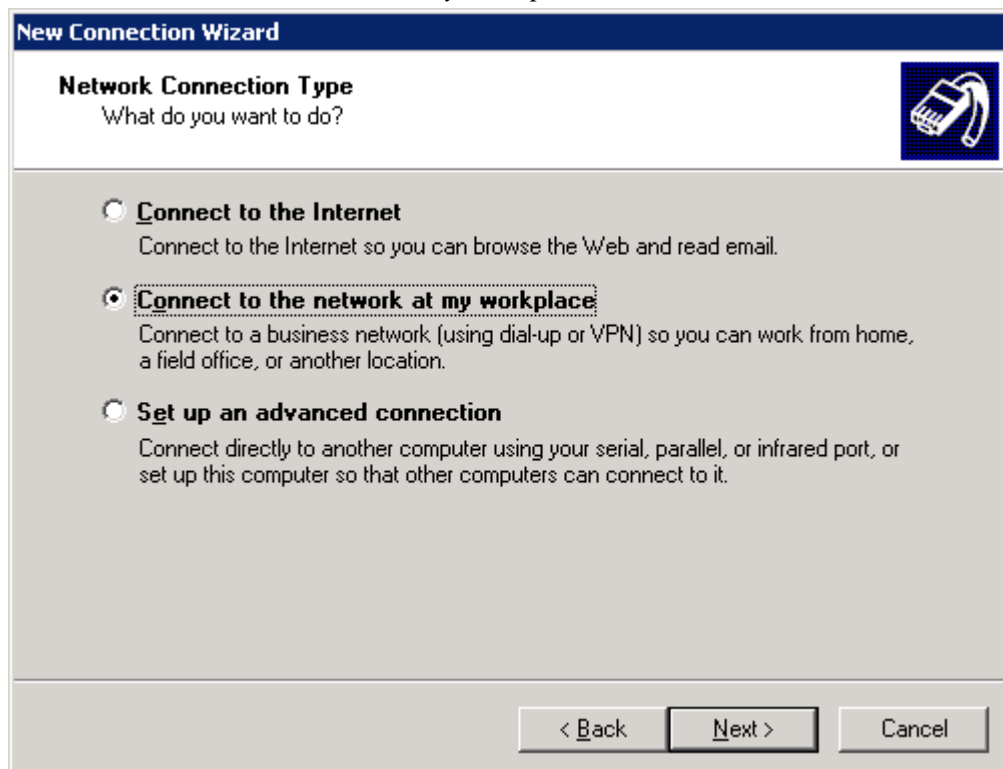
4. Click *Create a new connection* in the *Network Tasks* task pad:



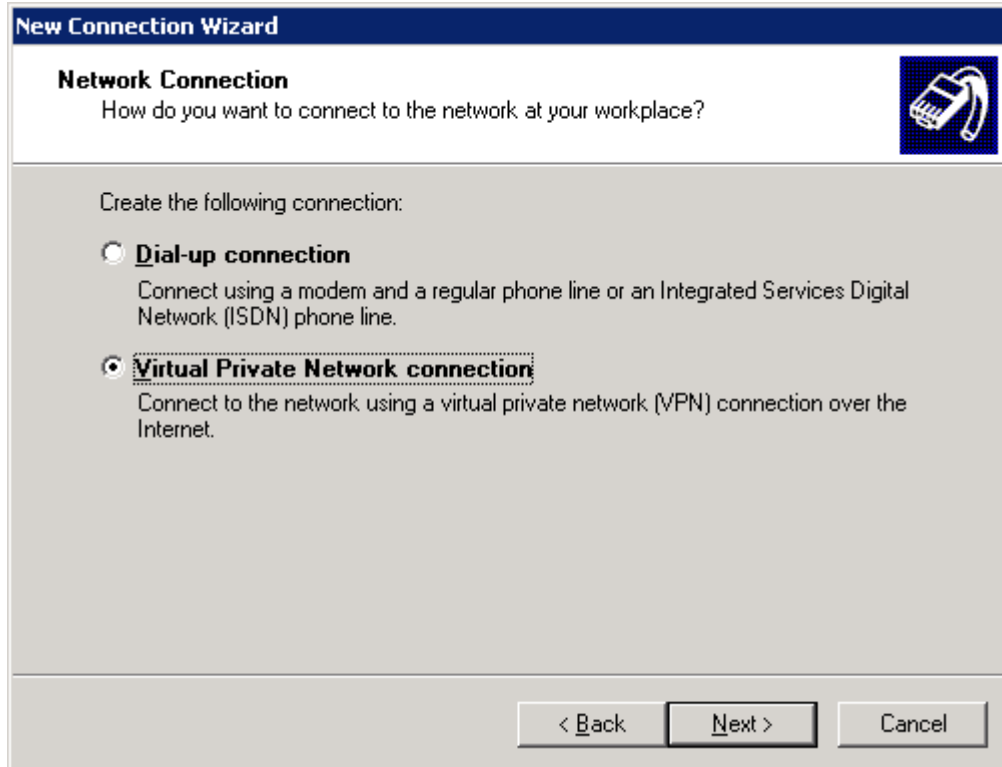
5. In the *Network Connection Wizard*, click *Next*:



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



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



The screenshot shows the 'New Connection Wizard' window with the 'Network Connection' tab selected. The title bar reads 'New Connection Wizard'. Below the title bar, the tab is labeled 'Network Connection' and the question 'How do you want to connect to the network at your workplace?' is displayed. To the right of the question is an icon of a network card. The main area contains the instruction 'Create the following connection:' followed by two radio button options. The first option is 'Dial-up connection' with the description 'Connect using a modem and a regular phone line or an Integrated Services Digital Network (ISDN) phone line.' The second option, 'Virtual Private Network connection', is selected and has a dotted border around it, with the description 'Connect to the network using a virtual private network (VPN) connection over the Internet.' At the bottom right are three buttons: '< Back', 'Next >', and 'Cancel'.

New Connection Wizard

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

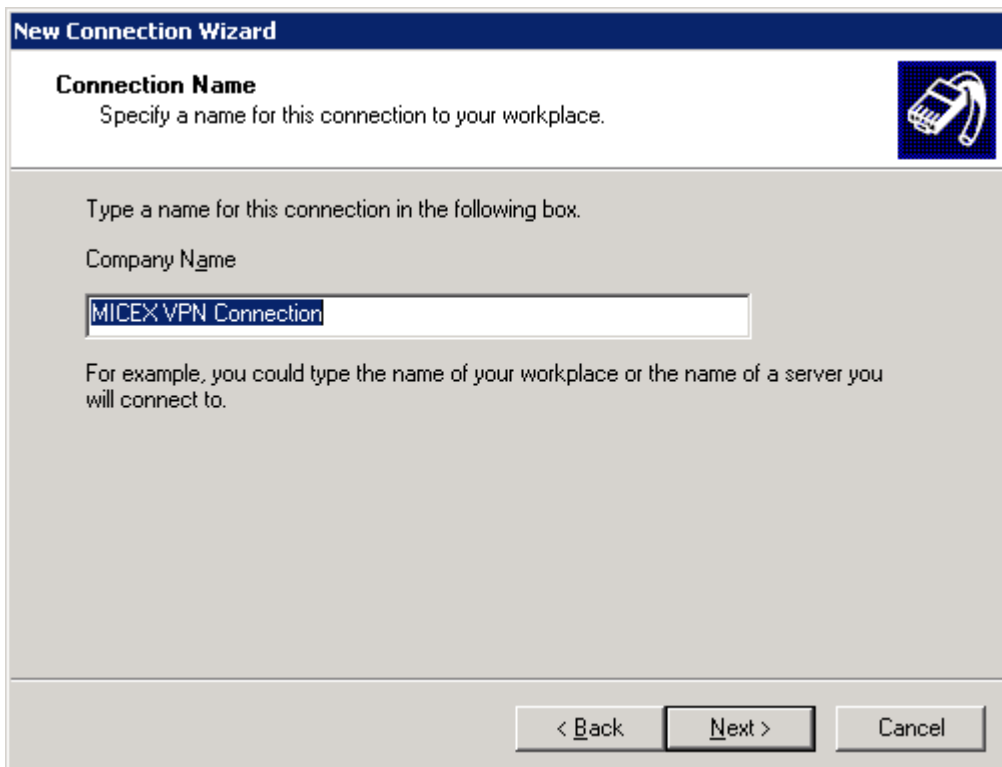
Create the following connection:

☐ **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.

< Back Next > Cancel

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



The screenshot shows the 'New Connection Wizard' window with the 'Connection Name' tab selected. The title bar reads 'New Connection Wizard'. Below the title bar, the tab is labeled 'Connection Name' and the instruction 'Specify a name for this connection to your workplace.' is displayed. To the right of the instruction is an icon of a network card. The main area contains the instruction 'Type a name for this connection in the following box.' followed by a text box labeled 'Company Name'. The text box contains the text 'MICEX VPN Connection'. Below the text box is a note: 'For example, you could type the name of your workplace or the name of a server you will connect to.' At the bottom right are three buttons: '< Back', 'Next >', and 'Cancel'.

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.

< Back Next > Cancel

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:

< Back Next > Cancel

10. Type the IP address, and then click *Next*:

New Connection Wizard

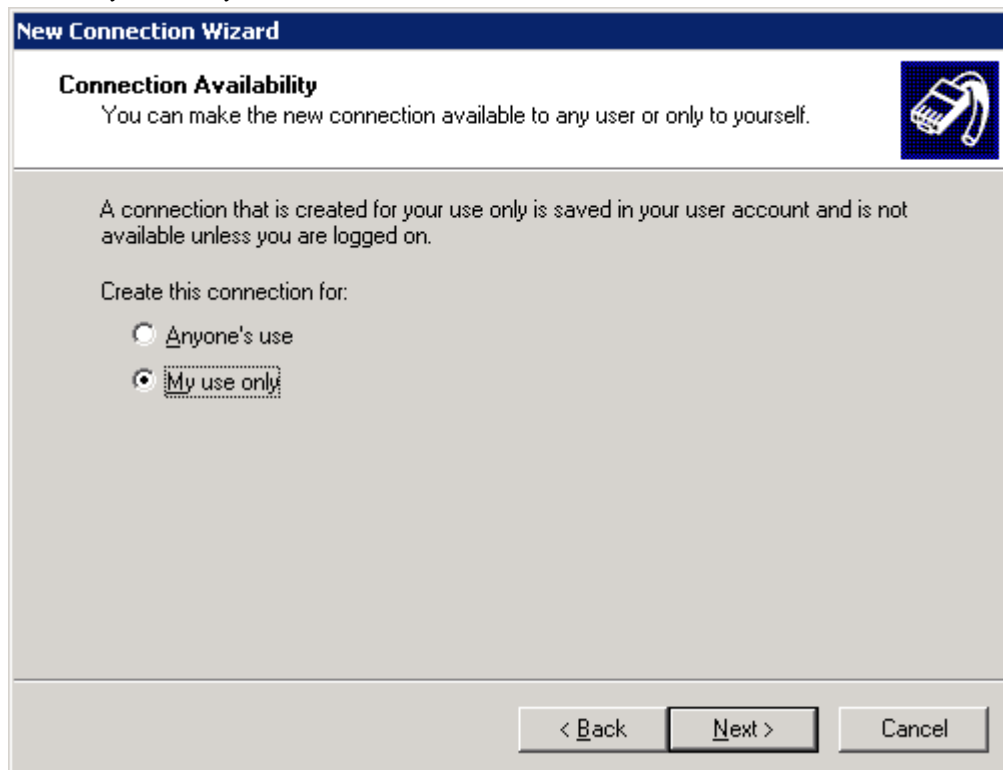
VPN Server Selection
What is the name or address of the VPN server?

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):

< Back Next > Cancel

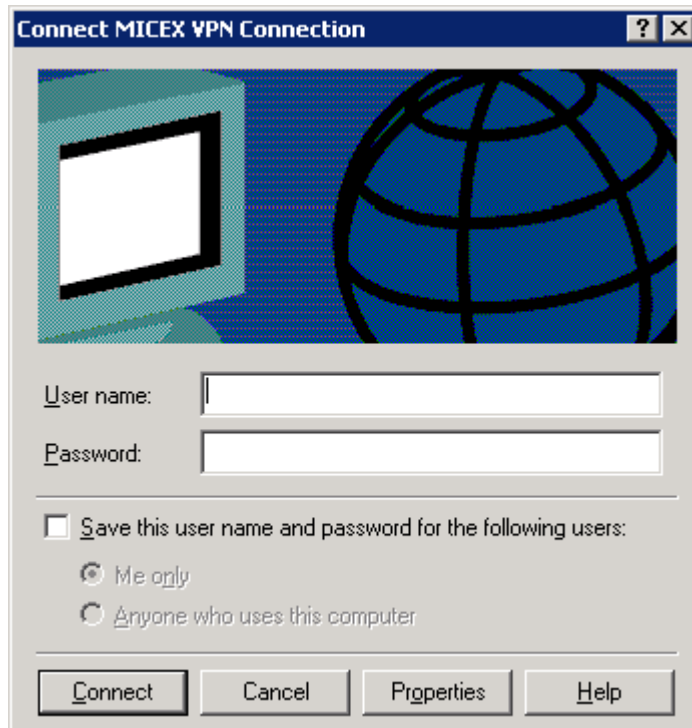
11. Click *My use only* and then *Next*:



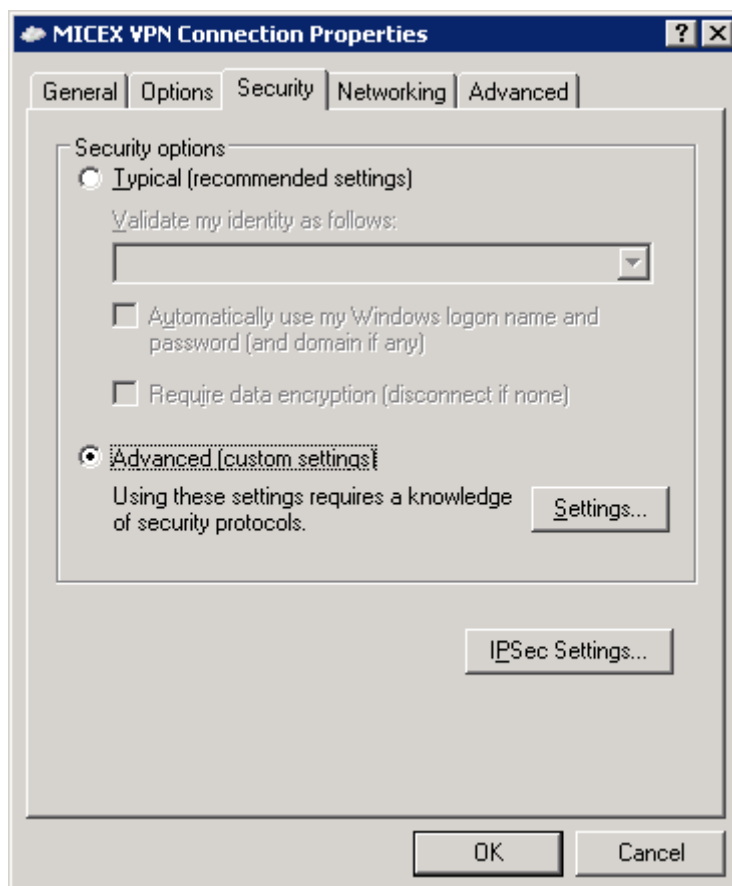
12. Click *Finish*:



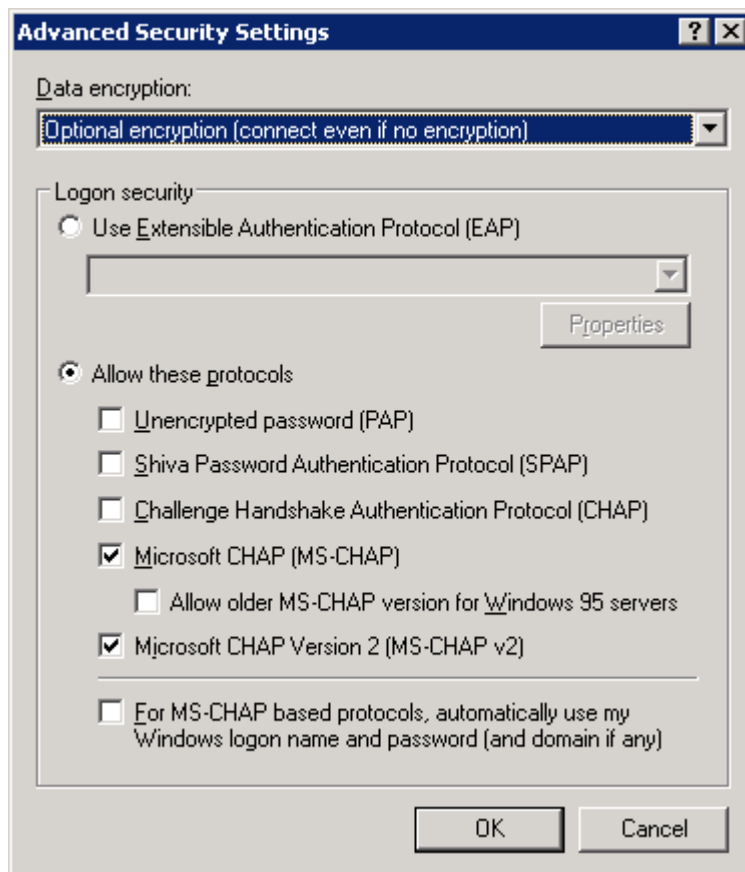
13. Leave *User name* and *Password* empty, and then click *Properties*



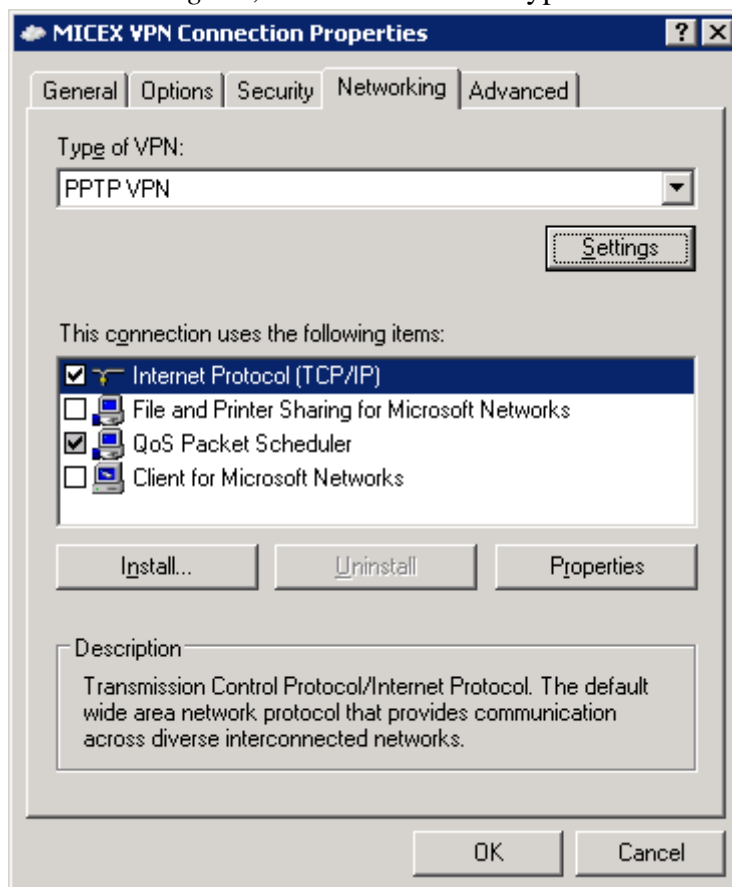
14. On *Security* tab, click *Advanced (custom settings)* and then *Settings...*:



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

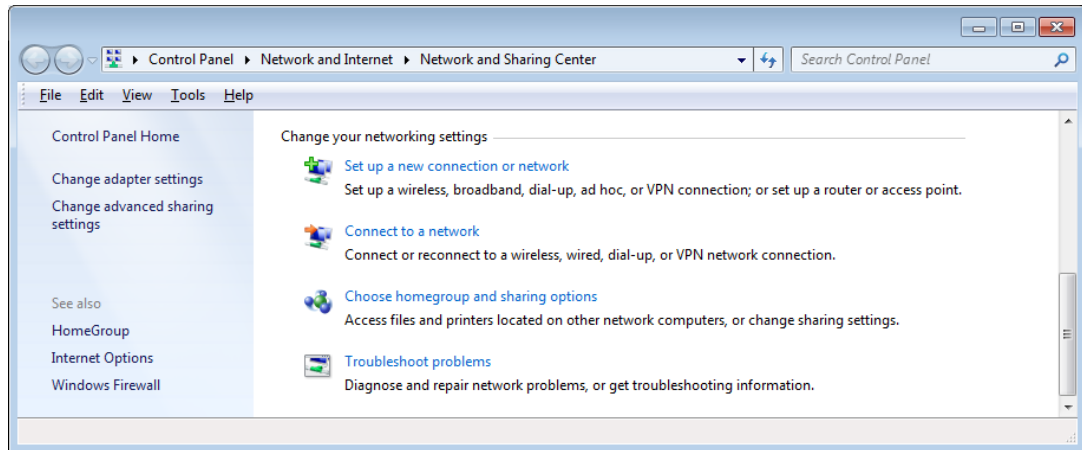


16. On *Networking* tab, choose *PPTP VPN* type of VPN and then click *OK*:

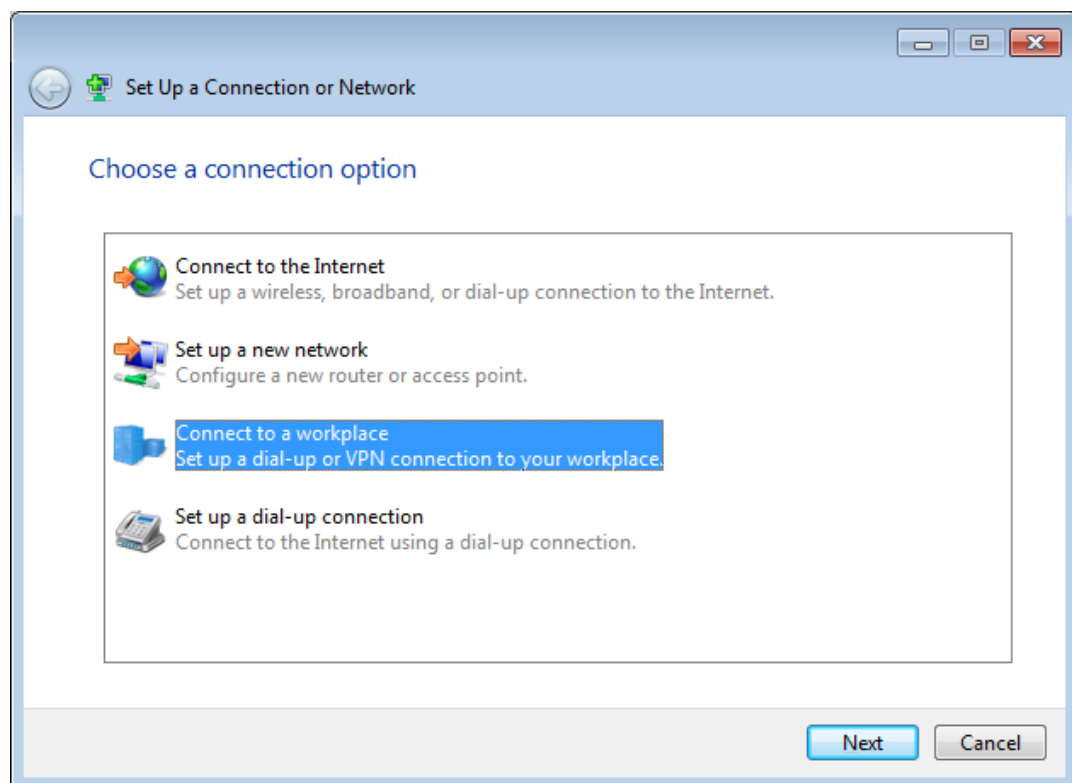


5.2. Configure a VPN connection with MICEX using Windows 7

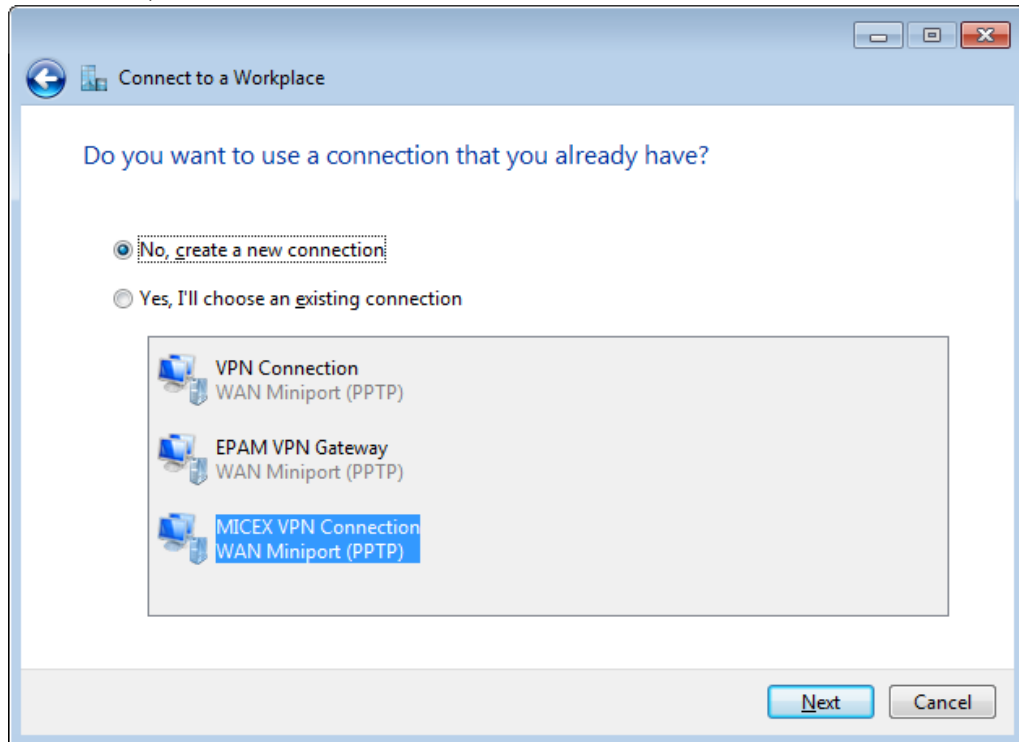
1. Make sure you are connected to the Internet
2. Open *Control Panel*→*Network and Internet*→*Network and Share Center* and then click *Set up a new connection or network*:



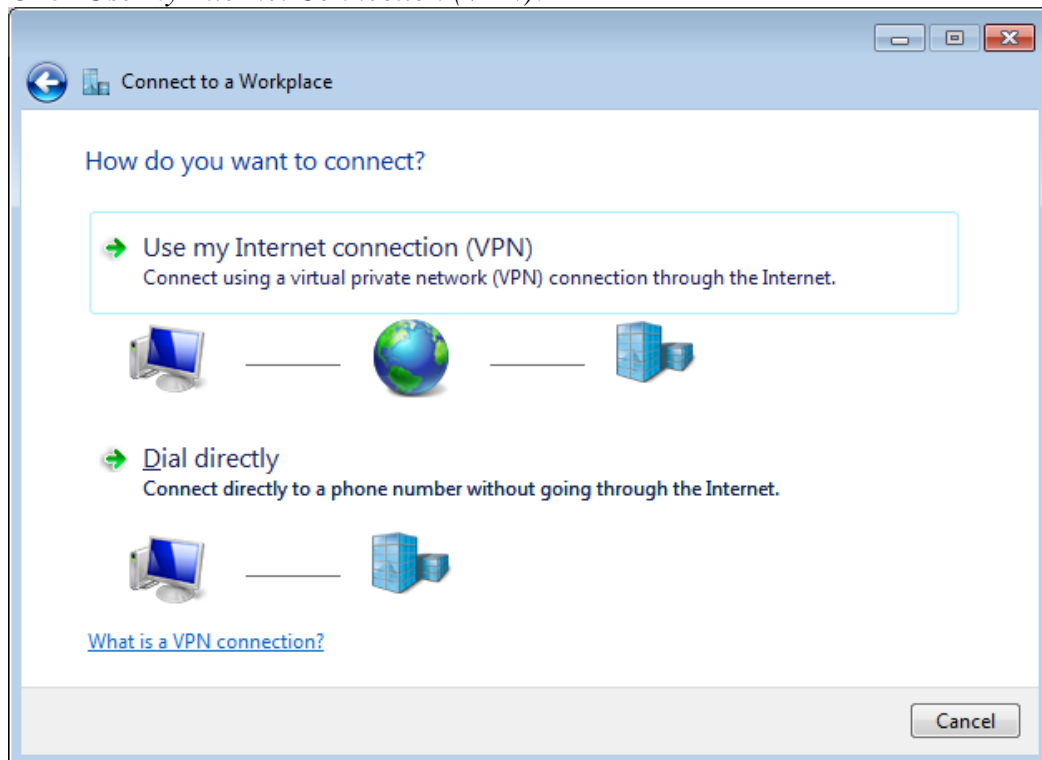
3. Choose *Connect to a workplace* and then click *OK*:



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

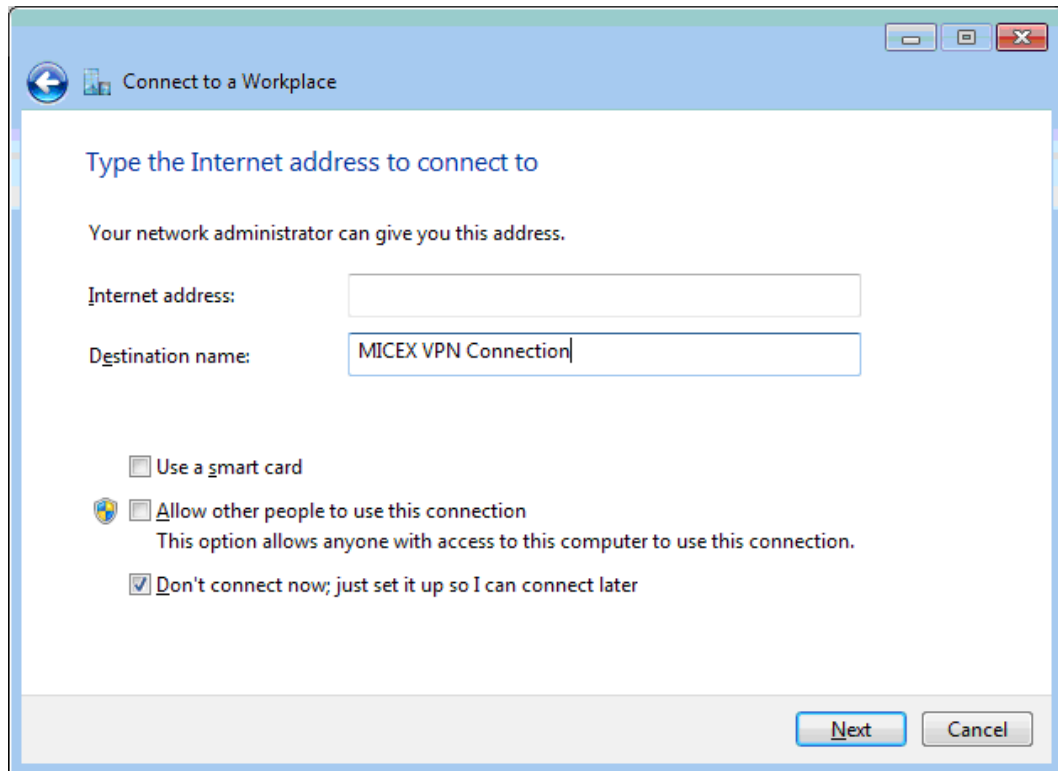


5. Click *Use my Internet Connection (VPN)*:



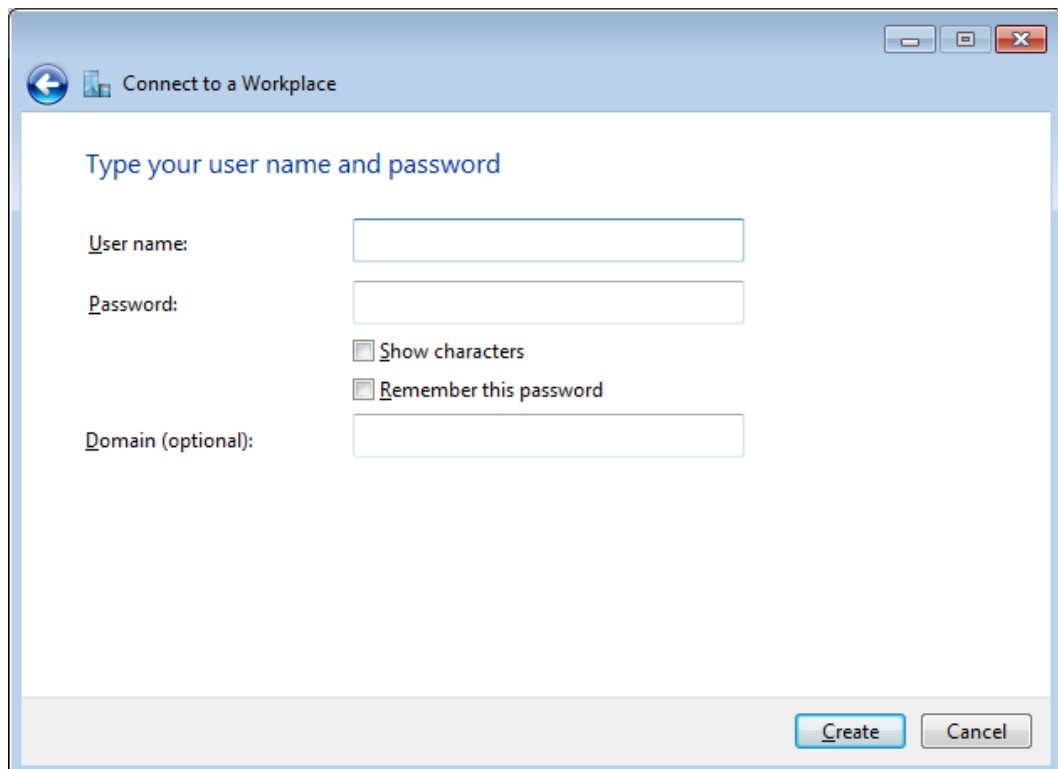
6. Type the IP address to the *Internet address* field, type MICEX VPN Connection to the *Destination name* field, check *Don't connect now; just set it up so I can connect later* and

then click *Next*:



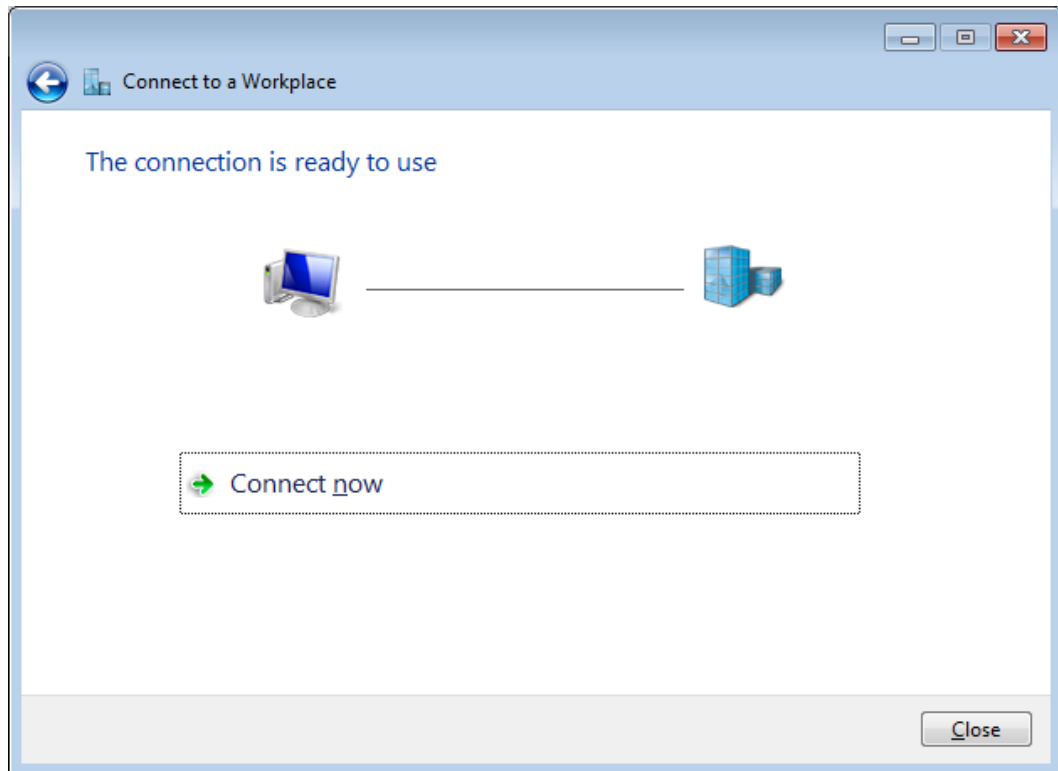
The screenshot shows a Windows-style dialog box titled "Connect to a Workplace". The main heading is "Type the Internet address to connect to". Below this, a message states: "Your network administrator can give you this address." There are two input fields: "Internet address:" which is empty, and "Destination name:" which contains the text "MICEX VPN Connection". Below the input fields are three checkboxes: "Use a smart card" (unchecked), "Allow other people to use this connection" (unchecked), and "Don't connect now; just set it up so I can connect later" (checked). A sub-note for the second checkbox reads: "This option allows anyone with access to this computer to use this connection." At the bottom right, there are two buttons: "Next" and "Cancel".

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

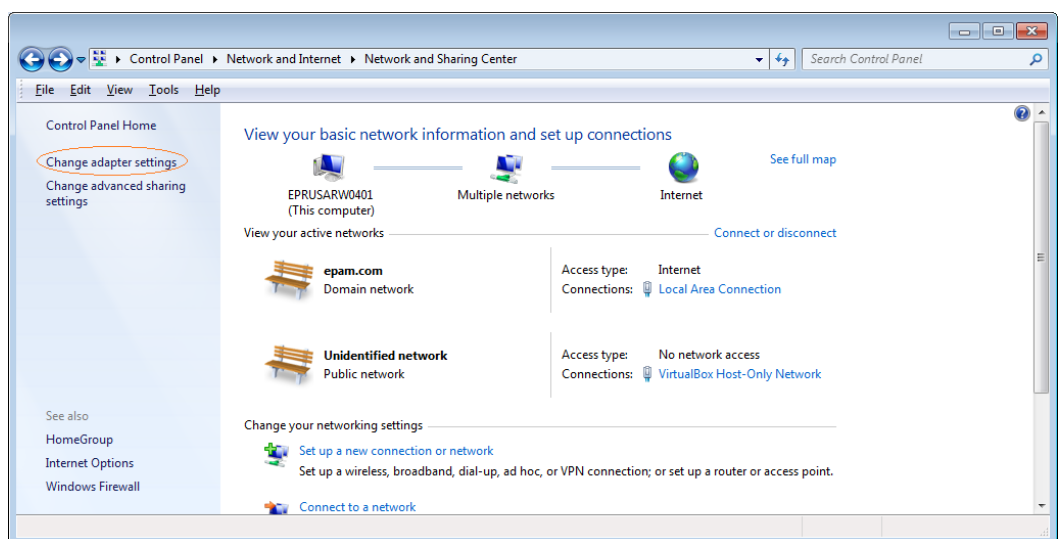


The screenshot shows the same "Connect to a Workplace" dialog box, but at a different step. The main heading is "Type your user name and password". There are three input fields: "User name:", "Password:", and "Domain (optional):", all of which are empty. Below the "Password:" field are two checkboxes: "Show characters" (unchecked) and "Remember this password" (unchecked). At the bottom right, there are two buttons: "Create" and "Cancel".

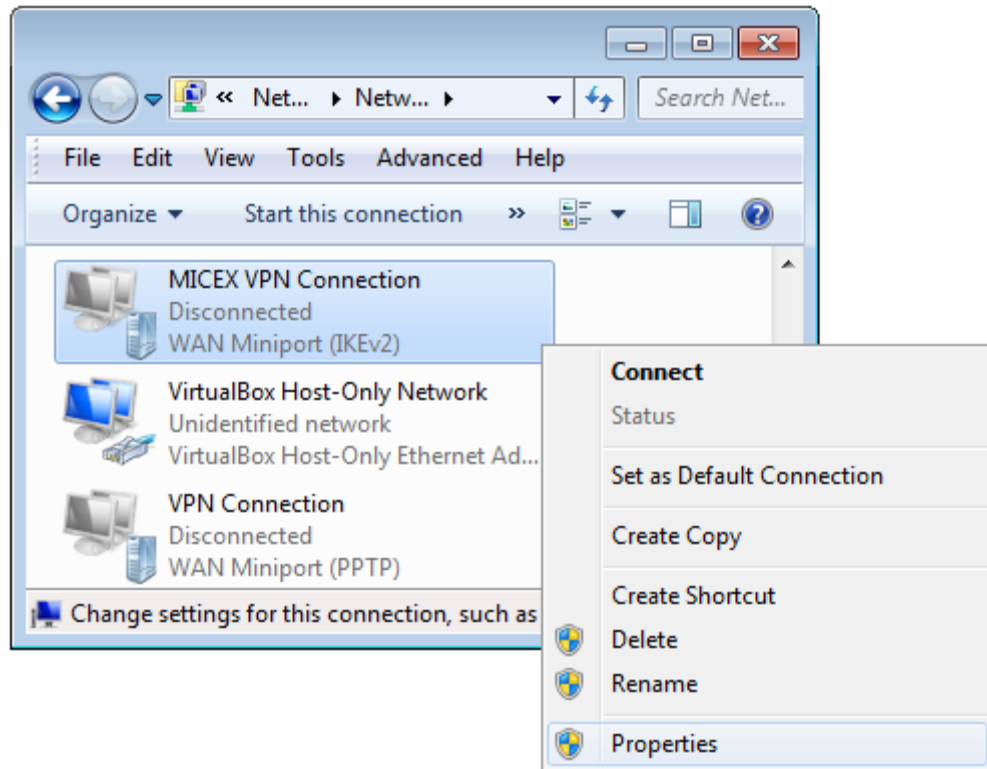
8. Click *Close*:



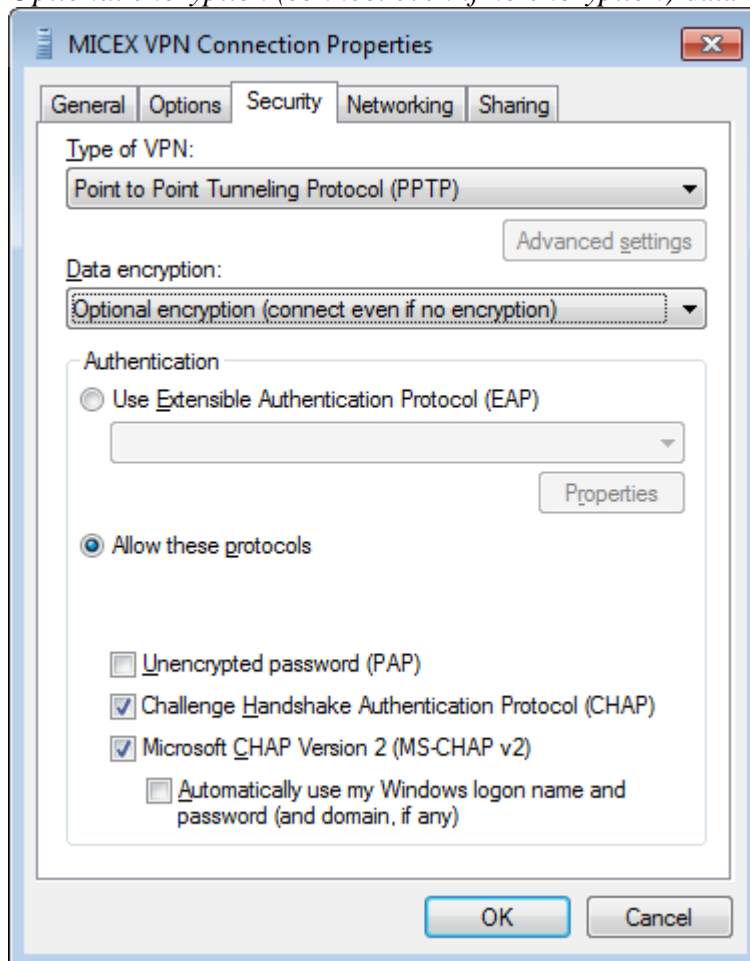
9. Open *Control Panel*→*Network and Internet*→*Network and Share Center* and click *Change adapter setting*:



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*:



5.3. Configure a VPN connection with MICEX 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 the IP-address and press enter:

```
Server IP: 95.222.333.45 + <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 all made right you will see:

```
Adding micex_vpn_connection - 95.222.333.45 - 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 configure 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: 95.222.333.45
# 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.

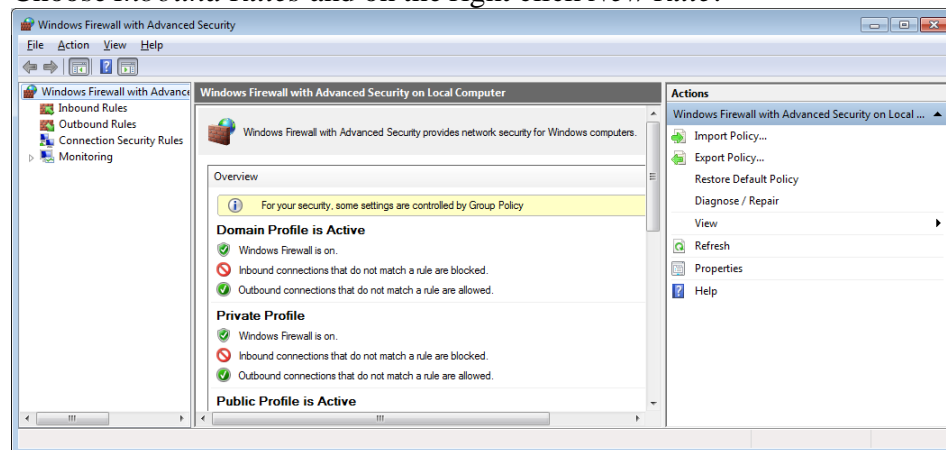
5.4.Troubleshooting

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

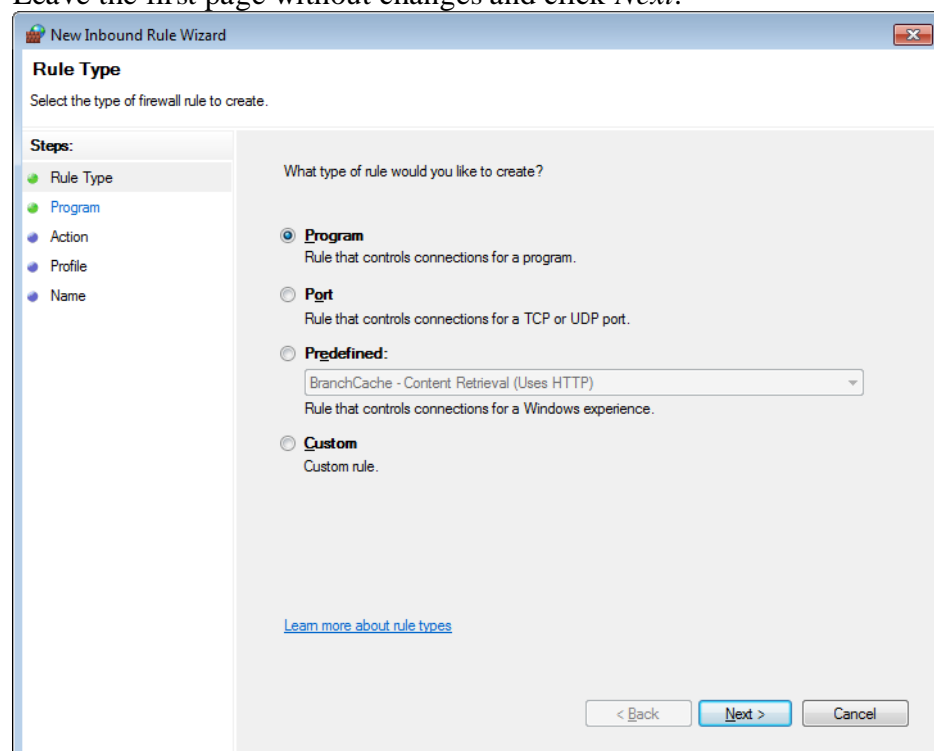
1.1 Open status of your VPN connection and check if the count of 'Received' bytes is continuously growing; If it's not so, ask for help the MICEX support team.

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:

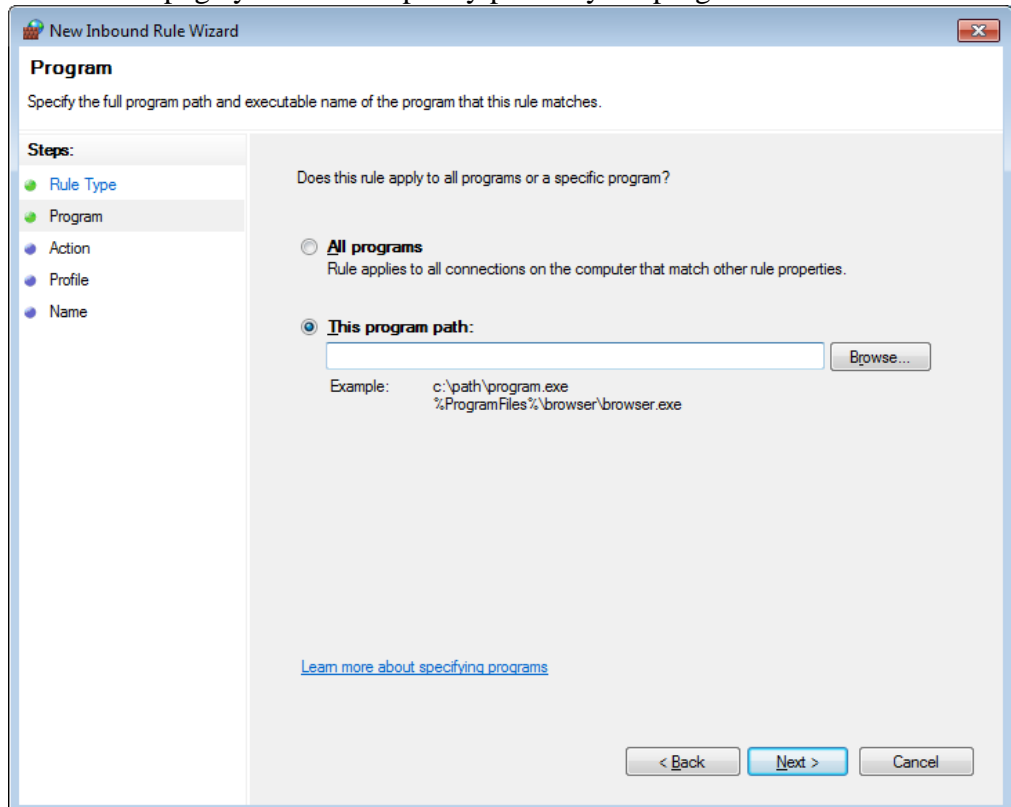
- ✓ Open *Windows Firewall*→*Advanced* settings;
- ✓ Choose *Inbound Rules* and on the right click *New Rule*:



- ✓ Leave the first page without changes and click *Next*:

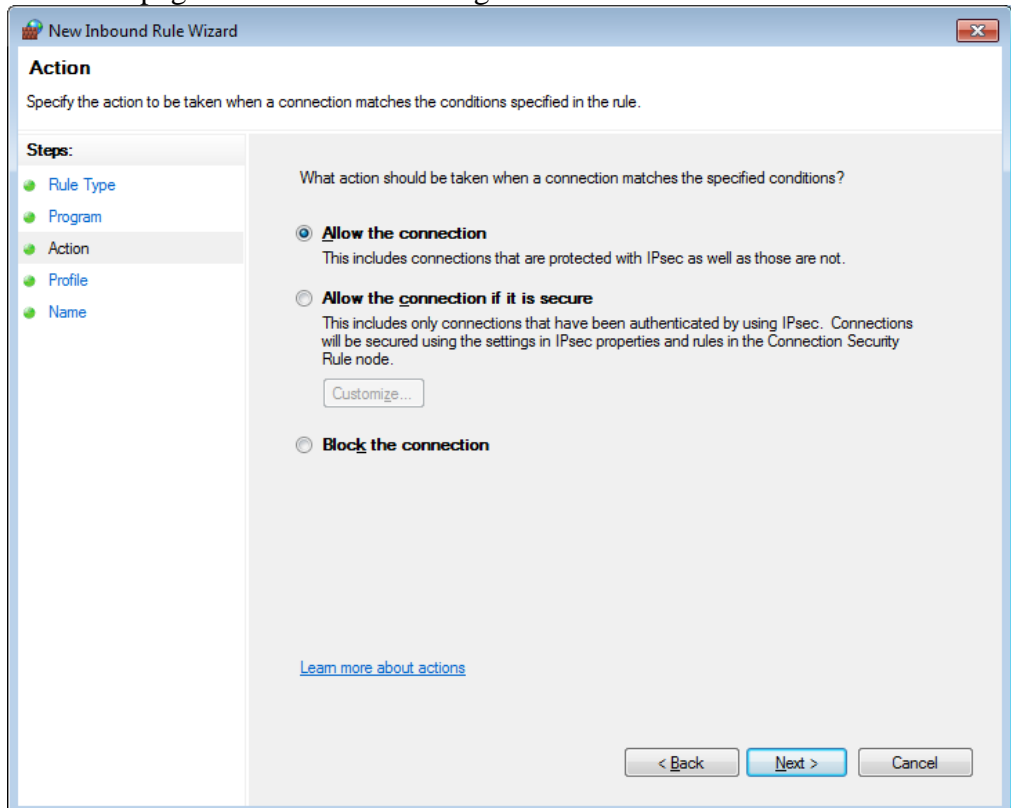


- ✓ On the next page you need to specify path to your program:

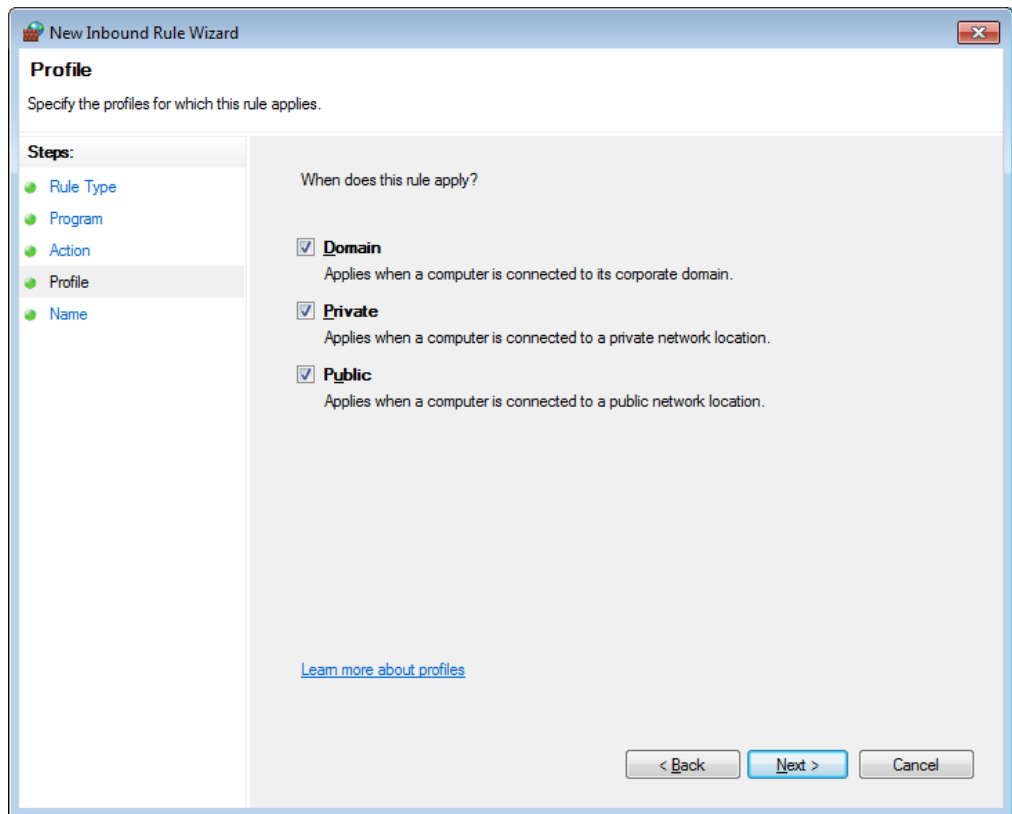


The screenshot shows the 'Program' step of the 'New Inbound Rule Wizard'. The title bar reads 'New Inbound Rule Wizard'. The main heading is 'Program'. Below it, the instruction says: 'Specify the full program path and executable name of the program that this rule matches.' On the left, a 'Steps:' list shows 'Rule Type', 'Program' (highlighted), 'Action', 'Profile', and 'Name'. The main area asks: 'Does this rule apply to all programs or a specific program?'. There are two radio button options: 'All programs' (unselected) and 'This program path:' (selected). The 'This program path:' option has a text input field and a 'Browse...' button. Below the input field, an example is provided: 'Example: c:\path\program.exe' and '%ProgramFiles%\browser\browser.exe'. At the bottom, there is a link 'Learn more about specifying programs' and three buttons: '< Back', 'Next >', and 'Cancel'.

- ✓ Leave the pages below without changes:



The screenshot shows the 'Action' step of the 'New Inbound Rule Wizard'. The title bar reads 'New Inbound Rule Wizard'. The main heading is 'Action'. Below it, the instruction says: 'Specify the action to be taken when a connection matches the conditions specified in the rule.' On the left, a 'Steps:' list shows 'Rule Type', 'Program', 'Action' (highlighted), 'Profile', and 'Name'. The main area asks: 'What action should be taken when a connection matches the specified conditions?'. There are three radio button options: 'Allow the connection' (selected), 'Allow the connection if it is secure' (unselected), and 'Block the connection' (unselected). The 'Allow the connection' option has a description: 'This includes connections that are protected with IPsec as well as those are not.' The 'Allow the connection if it is secure' option has a description: '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.' Below the 'Allow the connection if it is secure' option is a 'Customize...' button. At the bottom, there is a link 'Learn more about actions' and three buttons: '< Back', 'Next >', and 'Cancel'.



New Inbound Rule Wizard

Profile

Specify the profiles for which this rule applies.

Steps:

- Rule Type
- Program
- Action
- Profile
- Name

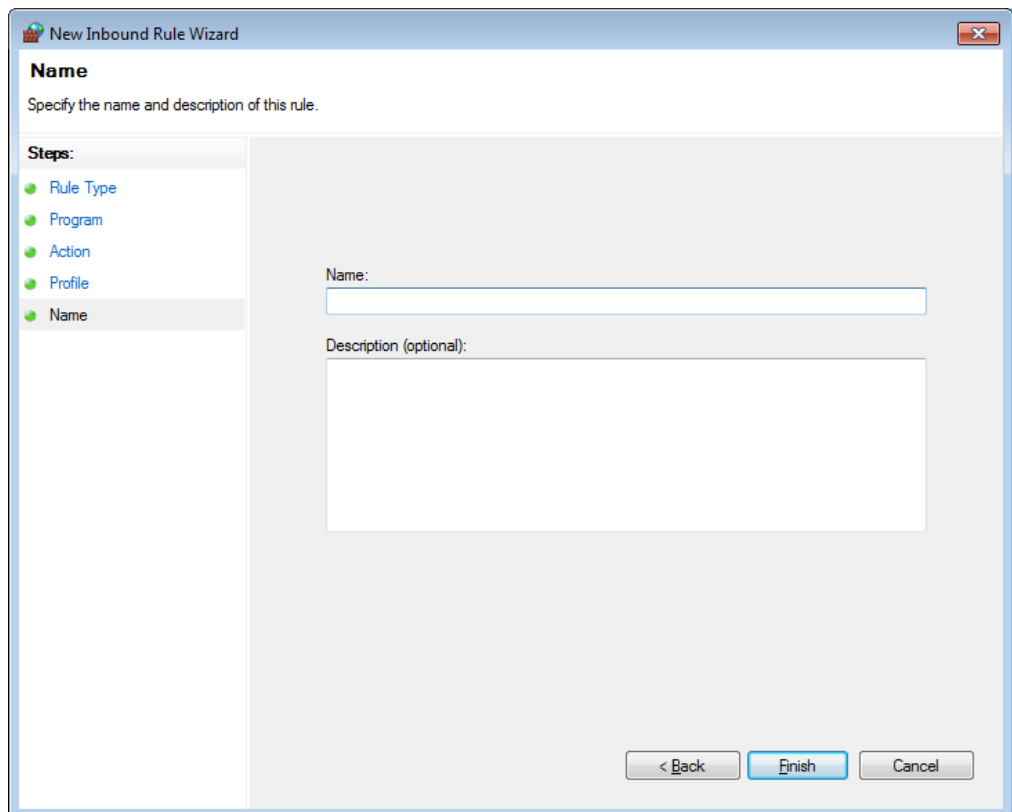
When does this rule apply?

- ☒ **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.

[Learn more about profiles](#)

< Back Next > Cancel

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



New Inbound Rule Wizard

Name

Specify the name and description of this rule.

Steps:

- Rule Type
- Program
- Action
- Profile
- Name

Name:

Description (optional):

< Back Finish Cancel

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 MICEX Market Data Multicast FIX/FAST Platform.

[FIX Antenna™](#) (C++ and .NET) allows one to transparently subscribe to Market Data from MICEX 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™ with MICEX Market Data Multicast FIX/FAST Platform. .NET package also contains the GUI client, which receives Market Data from MICEX 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 &nmsgs,
                               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,
                       instrument_listener_impl *ins_listener);

int main(int argc, char *argv[])
{
    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()->createMICEXApplication(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, |

| | |
|--------------|--|
| | <code>mfix_recovery_reason reason, mfix_feed_type feed_type)=0</code> |
| | 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::RefCountPtr`). 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::RefCountPtr). 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::RefCountPtr). Object must be created via "new" keyword only.