MTBT API Specification

Version: 6.1
Date: January 30, 2019

Revision History

Name	Description	Date
Version 6.1	Details about Load Balancer. FAQ on handling crossed orders scenario	January 29, 2019

NSE – MTBT Feed

1. Introduction

The NSE MTBT Feed is the Tick By Tick data feed of NSE. It disseminates information about orders and trades on a real time basis. The feed consists of series of sequenced variable length messages. NSE MTBT Feed data is sent to clients on a connection-less UDP Multicast.

2. Session Details

The Tick Data feed is available as separate Multicast Channels for separate Streams. The contracts are distributed across the streams. For e.g. Contracts 1-4 will be on stream 1, contracts 5-8 on stream 2 and so on.

A Data packet consists of a Header which has Stream ID and Sequence number fields. The sequence Number for the first message of the day for the stream will be sent as 1. After that, it will be incremented by 1 for each consecutive message for the Stream.

In case of switchover to DR site during the day, the sequence no. for the first message for each stream from DR site will be sent as 1. After that, it will be incremented by 1 for each consecutive message for the Stream.

Master information is available in file every end of day and can be downloaded any time.
Masters Information provides contract descriptor information about all the contracts available through different Streams and also provides the mapping of a contract to a particular Stream.

3. Data Type

Data Type	Size In Bytes
CHAR	1
SHORT	2
INT	4
LONG	8
DOUBLE	8

Byte order - Little Endian.
Pragma Pack – 1.

4. Packet Format

Server sends all the packets in following format:

Stream Header:

```
typedef struct
{
        short  msg_len;
        short  stream_id;
        int     seq_no;
}STREAM_HEADER;
```

Stream Data:

```
typedef struct
{
        char   cMsgType;
        .
        .
        .
}STREAM_DATA;
```

Each Data Packet consists of a Header and Data.

The Header (STREAM_HEADER) consists of following fields:

1. Message Length: This is the total size of the packet including Header and Data i.e. Sum of length of STREAM_HEADER and STREAM_DATA.
2. Stream Id : This identifies a particular stream
3. Sequence No : This is the sequence number of the packet for a particular Stream Id

Each Data Packet, ie. STREAM_DATA has Message Type as the first field. Always, the first field should be read first and interpreted. Depending on the value of this field the Data Packet should be mapped to relevant message structure.

5. Masters Data Details

To be able to interpret the data feed, the Client requires Masters Data.

The data contains contract information such as Token, Symbol, Instrument, Expiry Date, Strike Price and Option Type. This data also provides the mapping of Contract Information to Stream ID i.e. the Stream in which the contract is available.

All the Order and Trade Messages have only the Token Field for representing the contract. The Client Application must load the Masters Data for the day before receiving Orders and Trades. Otherwise the Client Application will be working on wrong information.

5.1. Contract Information File Details

This is a plain-text CSV file containing information for normal contracts for the trading day.

Nomenclature:
FO Segment:
fo_contract_stream_info.csv
CD Segment:
cd_contract_stream_info.csv
CM Segment:
cm_contract_stream_info.csv
CO Segment:
co_contract_stream_info.csv

File Format:
Plain Text, CSV

Field delimiter:
Comma (,)

Header Record:
This is the first line in the file. It contains the file generation timestamp along with number of contracts in the file, each followed by a comma.

Field Name	Data Type	Value	Remark
Date	INT	Numeric	File generation time in seconds from 01-Jan-1980 00:00:00
No. of contracts	INT	Numeric	Number of contract records

Contract Information Records:
The fields of the subsequent lines should be interpreted as follows. Each field is followed by a comma.

Field Name	Data Type	Value	Remark
Message Type	CHAR	'C'	Contract Information

Stream ID	SHORT	Numeric	Availability of this contract on a particular Stream
Token Number	INT	Numeric	Unique identifier for contract
Instrument	CHAR[6]	Alphanumeric	FO & CD: Security Instrument. Eg. FUTIDX CM: Set to "EQUITY". CO: Commodity Instrument. Eg. FUTENR, FUTBLN
Symbol	CHAR[10]	Alphanumeric	Security Symbol
Expiry Date	INT	Numeric	FO & CD: Expiry date of contract in seconds from 01-Jan-1980 00:00:00 CM: Not used, set to 0.
Strike Price	INT	Numeric	FO & CD: Strike Price Of Contract (In Paise). For FO segment this should be divided by 100 for converting into Rupees. For CD segment this should be divided by 10^7 for converting into Rupees. This will be zero in case of futures contract. CM: Not used, set to 0. CO: This will be zero in case of futures commodity contract.

Option Type	CHAR[2]	Alphanumeric	FO & CD Option Type for the contract CM: Series of the security. CO: Not used currently

5.2. Spread Contract Information File Details

Derivative segments only (FO, CD & CO):

This is a plain-text CSV file containing information for spread contracts for the trading day.

Nomenclature:
FO Segment:
fo_spd_contract_stream_info.csv
CD Segment:
cd_spd_contract_stream_info.csv
CO Segment:
co_spd_contract_stream_info.csv

File Format:
Plain Text, CSV

Field delimiter:
Comma (,)

Header Record:
This is the first line in the file. It contains the file generation timestamp along with number of contracts in the file, each followed by a comma.

Field Name	Data Type	Value	Remark
Date	INT	Numeric	File generation time in seconds from 01-Jan-1980 00:00:00
No. of spread contracts	INT	Numeric	Number of spread contract records

Spread Contract Information Records:
The fields of the subsequent lines should be interpreted as follows. Each field is followed by a comma.

Field Name	Data Type	Value	Remark
Message Type	CHAR	'P'	Spread Contract information

Stream ID	SHORT	Numeric	Availability of this spread contract on a particular Stream
Token 1	INT	Numeric	Token No. of First Spread Contract
Token 2	INT	Numeric	Token No. of Second Spread Contract

6. Sequenced Data Messages (Market Data)

Sequenced data messages will be sent by server on UDP Multicast and will contain market data. These messages will contain normal market, regular lot and spread data.

6.1. Order Message

For every new order request, modification request, cancellation request, this message is sent. All new order requests will be sent including the active orders. Modifications and cancellations must be handled at the client end on basis of Order Id and not on Price.

Currently Stop Loss Orders are not sent to clients. However when Stop Loss Order gets modified to Regular Lot Order or Regular Lot Order gets modified to Stop Loss Order it leads to following scenarios:
It is possible that for a Order Cancellation Message the original Order Id is not found by the Client Application. In that case the particular Order Cancel message should be ignored by the Client application
It is possible that for a Order Modification Message the original Order Id is not found by the Client Application. In that case the particular Order Modification Message should be treated as a New Order and handled accordingly.

Field Name	Data Type	Value	Remark

Global Header	STREAM_HEADER	Header Data	Refer "Global Header" definition in Section 4.
Message Type	CHAR	'N' or 'X' or 'M'	'N' - New Order 'M' - Order Modification 'X' - Order Cancellation
Timestamp	LONG	Numeric	Time in nanoseconds from 01-Jan-1980 00:00:00
Order ID	DOUBLE	Numeric	Day Unique Order Reference Number
Token	INT	Numeric	Unique Contract Identifier
Order Type	CHAR	Character	'B' - Buy Order 'S' - Sell Order
Price	INT	Numeric	Price of the order (In Paise) This field contains the price at which the order is placed. The price is in multiples of the tick size. For FO and CM segments this should be divided by 100 for converting into Rupees. For CD segment this should be divided by 10^7 for converting into Rupees. For CO segment, this should be divided by 10^4 for converting into rupees.
Quantity	INT	Numeric	Quantity of the order

6.2. Trade Message

This message is sent whenever an order in the order book gets executed fully or partially.

It is possible that either Buy Order ID or Sell Order ID can have value as Zero. In such a case that Order ID should be ignored and there should not be any follow up action based on that Order ID. Both the Order Ids will not have the value zero at the same time.

Field Name	Data Type	Value	Remark
Global Header	STREAM_HEADER	Header Data	Refer "Global Header" definition in Section 4.
Message Type	CHAR	'T'	'T' = Trade Message
Timestamp	LONG	Numeric	Time in nanoseconds from 01-Jan-1980 00:00:00
Buy Order ID	DOUBLE	Numeric	Day Unique Order Reference Number for Buy-Side Order
Sell Order ID	DOUBLE	Numeric	Day Unique Order Reference Number for Sell-Side Order
Token	INT	Numeric	Unique Contract Identifier
Trade Price	INT	Numeric	Trade Price (In Paise) This field contains the price at which the trade took place. The price is in multiples of the tick size. For FO and CM segments this should be divided by 100 for converting into Rupees. For CD segment this should be divided by 10^7 for converting into Rupees. For CO segment, this should be divided by 10^4 for converting into rupees.

Trade Quantity	INT	Numeric	Trade Quantity

6.3. Spread Order Message

FO and CD segments only:

For every new spread order request, modification request, cancellation request, this message is sent. All new spread order requests will be sent including the active orders.

Field Name	Data Type	Value	Remark
Global Header	STREAM_HEADER	Header Data	Refer "Global Header" definition in Section 4.
Message Type	CHAR	'G' or 'H' or 'J'	'G' - New Spread Order 'H' – Spread Order Modification 'J' – Spread Order Cancellation
Timestamp	LONG	Numeric	Time in nanoseconds from 01-Jan-1980 00:00:00
Order ID	DOUBLE	Numeric	Day Unique Order Reference Number
Token	INT	Numeric	Unique Contract Identifier
Order Type	CHAR	Character	'B' - Buy Order 'S' - Sell Order
Price	INT	Numeric	Price of the order (In Paise) This field contains the price difference for this spread. The price is in multiples of the tick size. For FO segment this should be divided by 100 for converting into Rupees.

			For CD segment this should be divided by 10^7 for converting into Rupees. For CO segment, this should be divided by 10^4 for converting into rupees.
Quantity	INT	Numeric	Quantity of the spread order

6.4. Spread Trade Message

FO and CD segments only:

This message is sent whenever a spread order in the order book gets executed fully or partially. It is possible that Buy Order ID or Sell Order ID can have value as Zero. In such a case that Order ID should be ignored and there should not be any follow up action based on that Order ID.

Field Name	Data Type	Value	Remark
Global Header	STREAM_HEADER	Header Data	Refer "Global Header" definition in Section 4.
Message Type	CHAR	'K'	'K' = Spread Trade Message
Timestamp	LONG	Numeric	Time in nanoseconds from 01-Jan-1980 00:00:00
Buy Order ID	DOUBLE	Numeric	Day Unique Order Reference Number for Buy-Side Order
Sell Order ID	DOUBLE	Numeric	Day Unique Order Reference Number for Sell-Side Order
Token	INT	Numeric	Unique Contract Identifier
Trade Price	INT	Numeric	Trade Price (In Paise)

			This field contains the price at which the trade took place. The price is in multiples of the tick size. For FO segment this should be divided by 100 for converting into Rupees. For CD segment this should be divided by 10^7 for converting into Rupees. For CO segment, this should be divided by 10^4 for converting into rupees.
Quantity	INT	Numeric	Quantity of the spread trade

7. Recovery Details

Recovery Server provides recovery of missed Tick Data for a particular multicast stream.

If the end client application misses any tick data it can recover the ticks by requesting the Recovery Server for the missed tick. This request is on TCP. The Recovery Server will provide the missed tick data on TCP.

7.1. Tick Data Recovery Request Message

Whenever tick data is missed this message needs to be sent to Recovery Server on TCP. No header is required for this message. In case only one tick is missed Start Sequence No. and End Sequence No. should be same.

A Tick Data Recovery Response Message is sent first by the Recovery Server to indicate whether this request will be processed successfully by Recovery Server.
If the Tick Recovery is successful then rest of the response messages are provided on TCP and format is same as mentioned in section 4.

If the Tick Recovery Request is unsuccessful, Member Application can send the request again.

Field Name	Data Type	Value	Remark
Message Type	CHAR	'R'	Recovery Request
Stream ID	SHORT	Numeric	Unique Stream Reference
Start Sequence Number	INT	Numeric	Requested Message Sequence Number
End Sequence Number	INT	Numeric	Requested Message Sequence Number

7.2. Tick Data Recovery Response Message

This is the first message sent as response by Recovery Server on TCP to indicate whether Tick Data recovery request has been processed successfully by Recovery Server.
For this message, the sequence number field in Global Header will have the value 0 (zero).

Field Name	Data Type	Value	Remark
Global Header	STREAM_HEADER	Header Data	Refer "Global Header" definition in Section 4.

Message Type	CHAR	'Y'	Recovery Response
Request Status	CHAR	'S' or 'E'	'S' – Success 'E' – Error

8. Heartbeat Message

Heartbeat message will be sent when there is no data available for a few seconds. These messages will be available on all Streams.

In case of Heartbeat messages, the sequence number field in Global Header will have the value 0 (zero).

Field Name	Data Type	Value	Remark
Global Header	STREAM_HEADER	Header Data	Refer "Global Header" definition in Section 4.
Message Type	CHAR	'Z'	Heartbeat message
Last Sequence No.	INT	Numeric	Last sent data sequence no. of the Stream.

9. Miscellaneous

1. Old Price and Quantity is not being provided as part of Order Modification and Cancellation. How to handle Order Modification and Cancellation?

All Order Messages are provided with Order Id data. Tracking of Orders needs to be done by Member Applications based on Order id. Whenever an Order Modification message is received the previous Order Image needs to be looked based up on Order Id and accordingly adjusted with the Modified Order Message.
Same behaviour for Order Cancellation Message.

2. Are active or aggressive Orders being sent as part of Order Messages?

Yes, active or aggressive orders are also being sent as part of Order Messages.

3. **Is there any specific handling required for DQ functionality?**

DQ functionality will be handled at Exchange End. For Member End Applications there is no specific handling required.

4. **The multicast feed specifications are applicable for which market segments?**

The specifications are applicable for NSE CM, FO, CD & CO market segments.

5. **Is Order Id same as interactive Order Number?**

Yes, Order id is same as interactive Order Number.

6. **Trade Message has an Order Id which is not present in client Order Book. Is this scenario possible?**

Yes, this scenario is possible as market orders are not being sent. In such scenario the particular Order Id should be ignored by the client application.

7. **Order Cancel message has a price or quantity which is different from the original order. Is this possible?**

Yes, this scenario is possible. For Order Cancel messages the client application should consider only the Order Id field and accordingly remove the original order from the order book.

8. **Is there a limit to the maximum number of messages which can be recovered in a single session from the Exchange Recovery Servers?**

Yes. This is currently 300000 messages. More messages can be recovered through subsequent requests.

9. Is there any handling required for pre-open orders in CM segment?

No specific handling is required for pre-open orders at the Member end. Pre-open orders will not be sent out on the MTBT Feed in the Order Collection phase of Pre-open Session. The Trades occurring in the Matching Phase of the Pre-open Session will be sent out on the feed. The Pending Orders which will be carried forward to the Normal Market Session will be sent out as normal orders after the Pre-open Session ends.

10. What is the use of Last Sequence No field in the Heartbeat packet?

Last Sequence No field in the heartbeat message can be used to derive the sequence number of subsequent message.

11. Is Matched or Crossed Orders Scenario possible?

Subscribers of MTBT feed can construct the order book based on the order and trade messages which are sent. As part of the feed, new orders including the active orders are sent. As active orders can result in trade, there is a possibility of matched or crossed orders being in the order book which is constructed by the subscribers (end client applications). However, the end client applications can detect this condition of matched or crossed orders (i.e. the order has resulted in trade) by doing a simple check or validation.

Based on every message received, the end client application needs to update the order book first. After updating the order book, the best buy price and best sell price will be available in the order book. If the best buy price is greater than or equal to the best sell price, then there are matched or crossed orders in the book i.e. Trades are taking place. By carrying out this check (best buy >= best sell), the scenario of matched or crossed orders can be detected.

10. Tuning guidelines

10.1. NIC Recommendations

The minimum value of the Max Receive Buffer size on the NICs which are going to be used for receiving the TBT Multicast should be 4 KB. NICs with the specified value below the recommended value may not be able to handle the incoming data bursts, resulting in UDP packet drops. In case of teamed NICs (NIC Bonds), this is applicable for both the underlying physical NICs.

Where: This is meant for the machine which runs Member application which subscribes to the TBT Multicast.

10.2. Increase the Kernel Receiver Backlog

To find out the current value, as root, execute the following command:

```
sysctl -a | grep netdev_max_backlog
```

```
/tbtdv1/users/tbtdv> sysctl -a | grep netdev_max_backlog
net.core.netdev_max_backlog = 1000
```

Make a note of the value of this parameter.

If it is lower than 50000, as root, execute the following command to increase it as follows:

```
/sbin/sysctl -w sys.net.core.netdev_max_backlog=50000
```

Verify whether the parameter has been set as desired:

```
/tbtdv1/users/tbtdv> sysctl -a | grep netdev_max_backlog
net.core.netdev_max_backlog = 50000
```

Where: This should be done on the machine which runs Member application which subscribes to the TBT Multicast.

10.3. Increase the NIC Ring Buffer Receive size on NICs:

As root, execute the following command for all of the NICs on the system which are used for receiving TBT Multicast data:

```
ethtool -g ethX
```

.. Where X corresponds to the NIC number (run the "ifconfig" command to list NICs on your system).

```
[idxmgr@indexdev1 ~]$ ethtool -g eth0
Ring parameters for eth0:
Pre-set maximums:
RX:             4078
RX Mini:        0
RX Jumbo:       0
TX:             511
Current hardware settings:
RX:             200
RX Mini:        0
RX Jumbo:       0
TX:             511
```

Make a note of the following values:

a) "RX" parameter displayed in the "Pre-set maximums" (maximum receive buffer size which can be set).
b) "RX" parameter displayed in the "Current hardware settings" (current receive buffer size).

If "b" is lower than "a", as root, execute the following command to increase such that it is set to the max:

`ethtool -G ethX rx a`

.. Where X corresponds to the NIC number and 'a' corresponds to maximum value that can be set for the RX parameter.

In case of teamed NICs (NIC Bonds), this tuning should be performed for the underlying physical NICs.

Verify whether the parameter has been set as desired:

```
[idxmgr@indexdev1 ~]$ ethtool -g eth0
Ring parameters for eth0:
Pre-set maximums:
RX:             4078
RX Mini:        0
RX Jumbo:       0
TX:             511
Current hardware settings:
RX:             4078
RX Mini:        0
RX Jumbo:       0
TX:             511
```

Where: This should be done on the machine which runs the Member application which subscribes to the TBT Multicast.

10.4. Increase the OS Send and Receive Buffers

As root, execute the following commands:

```
sysctl -a | grep rmem_max
```

```
/tbtdv1/users/tbtdv> sysctl -a | grep rmem_max
net.core.rmem_max = 4669986
```

```
sysctl -a | grep tcp_mem
```

```
/tbtdv1/users/tbtdv> sysctl -a | grep tcp_mem
net.ipv4.tcp_mem = 196608  262144  4669986
```

Make a note of the value of these parameters.

If the parameter rmem_max and third parameter of tcp_mem are lower than 134217728, as root, execute the following command to increase them to 134217728 accordingly:

```
/sbin/sysctl -w net.core.rmem_max=134217728
/sbin/sysctl -w net.ipv4.tcp_mem=10240 87380 134217728
```

Verify whether the parameter has been set as desired:

```
/tbtdv1/users/tbtdv> sysctl -a | grep rmem_max
net.core.rmem_max = 134217728
/tbtdv1/users/tbtdv> sysctl -a | grep tcp_mem
net.ipv4.tcp_mem = 196608  262144  134217728
```

(The first two values of tcp_mem shown above may differ with Member's machine.)

System-specific consideration:

The network read buffer of OS is being set to a significantly high value here (134 MB). This will become applicable system-wide, which means that all applications running on the system will be able to request this amount of OS network read buffer size from OS. Tune as per your available RAM.

Where: This should be done on the machine which runs the Member application which subscribes to the TBT Multicast.

10.5. Increase the Application Send and Receive Buffers

Ensure that large network Receive Buffer is requested from the OS through the Member application code on all the sockets used to receive UDP traffic.

```
int nTCPRecvBufSize = 134217728;
if(0 != setsockopt(nFD, SOL_SOCKET, SO_RCVBUF, &nTCPRecvBufSize,
sizeof(nTCPRecvBufSize))
{
     // This means that the call did not go through.
```

```
        // Log this error and take necessary action.
}
```

Where: This must be done in the Member application just after creation of the socket(s) used to receive the TBT Multicast.

10.6. Ensure that Multicast IP is also provided to "bind"

In Member software, when populating the "sockaddr_in" structure which is passed to "bind" system call, ensure that the Multicast IP Address is populated as well when populating the port. On systems which subscribe to multicast from both markets on same machine, this will result in multicast from both the markets to be received on the same socket, resulting in data issues.

```
int                     nRetVal = 0;
struct sockaddr_in      stLocalAddr;
memset(&stLocalAddr, 0, sizeof(stLocalAddr));

stLocalAddr.sin_family          = AF_INET;
stLocalAddr.sin_port            = htons(pstStreamAttr->nMulticastPort);
stLocalAddr.sin_addr.s_addr     = inet_addr(pstStreamAttr->szMulticastIP);

nRetVal = bind(pstStreamAttr->nRecvSockFD, (struct sockaddr *)&stLocalAddr,
                sizeof(stLocalAddr));
if(nRetVal != 0)
{
        nRetVal = errno;
        LOG_FATAL("Error in creating client socket <%s>", strerror(nRetVal));
        return FAILURE;
}
```

Where: This must be done in Member application when it binds to the Multicast Port before adding the Membership Request for receiving the TBT Multicast.

10.7. Disclaimer

The methods provided in the Tuning Guidelines are in an effort to try and help Members tune their systems. Members must take into careful consideration the load on their systems, availability of system resources and their specific use-cases before embarking on any system tuning exercise. The Exchange does not officially mandate the use of any of the tuning methods mentioned here and is not responsible for the results of any tuning exercise carried out by the Member at their end. Currently these guidelines are available for Linux systems only.

11. Recovery Guideline

The Tick Data feed is available on two separate Multicast Channels (Source 1 and Source 2) for each Stream on active-active basis. On one multicast channel the feed is delayed and lagging behind the other feed. Please refer the circular for details regarding the Source 1 and Source 2 multicast IP and port.

It is recommended that the end user application should subscribe to both multicast channels for each stream and receives the data from both the channels in active-active manner.
This needs to be done to ensure that data loss is minimized for the receiving applications as there is a possibility of data loss in multicast protocol.

If the end user application misses any tick data from both the active-active channels it can recover the ticks by sending recovery requests to recovery server on TCP.

The recovery request sent by client will be received by HA-Proxy load balancer at exchange side. The load balancer is used to distribute the recovery request to multiple recovery servers based on least number of connections algorithm. The load balancer will forward the recovery request to least connections recovery server. There are two HA-Proxy load balancers running at exchange side with single Virtual IP and Port. If one Load balancer goes down then recovery requests will be automatically forwarded to the other load balancer.

1. Client can make 13 connections concurrently from same IP address to Load Balancer for each market segment.

2. Client can connect 100 times per second from same IP address to Load Balancer. There should be an interval of at least 10 milliseconds between successive recovery requests.

3. After successful client connection with Recovery Server, client has to send the recovery request within one second otherwise Recover Server will disconnect the client without sending any response.