Shanghai Futures Exchange

API Specification for SHFE Market Data Platform 2.0

Version: 1.22
Released on: 2026/07/21

上海期貨交易所
SHANGHAI FUTURES EXCHANGE

Records of Revision, Verification and Review

Record of Revision

Version No.	Revised on	Summary
0.90	2018/09/25	Draft for comments finished
1.00	2018/11/28	Revised according to revision suggestions and released
1.10	2019/02/14	Added with information on instrument quotation update: In Section 5.2.4, four members including update time and instrument snapshot number, are added to the trade quotation field. Adjusted with multicast heartbeat: In Section 6.2.1, heartbeat interval is changed from 5 seconds to 3 seconds. Changed and added with some explanations: In Section 6.2.2, the incremental packet is changed to incremental message to distinguish it as one type of the messages of market data incremental; In Section 5.2.2, explanation is added to clarify that the response information field is the only field in the query failure feedback message; In Section 5.2.2, explanation is added about user-end product information and interface-end product information. Mistakes corrected: For example, the end mark in the snap time format of the snapshot time field in Section 5.2.4 is changed from '\n' to '\0'.
1.20	2019/04/18	Added with explanation on members related to index in the instrument information field: In Section 5.2.4, explanation on index instruments is added to the Productclass and CurrencyID members in the instrument information field.
1.21	2019/12/09	Added with explanation on members related to TAS in the instrument information field: In Section 5.2.4, the product type - TAS instrument is added to the Productclass in the instrument information field; the explanation on UnderlyingInstrID is changed from “Underlying options instrument code" to “Underlying instrument code".
1.22	2026/07/21	Added with explanation on members related to Spread in the instrument information field: In Section 5.2.4, the product type - Spread instrument is added to the Productclass in the instrument information field. Updated the Error Code List in Section 5.2.2.

Record of Verification

Verified by	Department (Unit)	Verified on
Wang Junjie	SHFE	2026/07/21

Record of Review

Reviewed by	Department (Unit)	Reviewed on
Jiang Peng	SFIT	2026/07/21

This document is made and maintained by: Shanghai Futures Information Technology Co., Ltd. (SFIT)

Contents

- 1. Preface
- 2. Market Data and Its Organization Structure
  - 2.1 Topic
  - 2.2 Market Data Snapshot and Market Data Incremental
- 3. Services
- 4. Member Type and Packet
  - 4.1 Member Type
  - 4.2 Packet
    - 4.2.1 Packet Structure
    - 4.2.2 Field Composition Grammar
    - 4.2.3 Message and Packet
- 5. Market Data Query Service
  - 5.1 MDQP Packet Header
  - 5.2 Functions
    - 5.2.1 Heartbeat
      - Heartbeat Message (TypeID=0x00)
    - 5.2.2 User Login
      - Login Request (TypeID=0x11)
      - Login Response (TypeID=0x12)
    - 5.2.3 User Logout
      - Logout Request (TypeID=0x13)
      - Logout Response (TypeID=0x14)
    - 5.2.4 Topic Snapshot Query
      - Snapshot Query Request (TypeID=0x31)
      - Snapshot Query Response (TypeID=0x32)
    - 5.2.5 Topic Incremental Query
      - Incremental Query Request (TypeID=0x33)
      - Incremental Query Response (TypeID=0x34)
- 6. Market Data Incremental Service
  - 6.1 MIRP Packet Header
  - 6.2 Functions
    - 6.2.1 Heartbeat
      - Heartbeat Message (TypeID=0x00)
    - 6.2.2 Incremental Feed
      - Incremental Refresh Message (TypeID=0x01)
- 7. Starting and Recovery
  - 7.1 Starting
  - 7.2 Market Data Recovery
    - 7.2.1 Recovery via Snapshot
    - 7.2.2 Recovery via Incremental Supplement
  - 7.3 Data Center Changing
- 8. Closing Words

1. Preface

Shanghai Futures Exchange (SHFE) has developed the Market Data Platform 2.0 (SMDP2.0). Supported by technical means such as data encoding & compression and multicast transmission, the Platform is responsible for sending trading data on futures and derivatives.

This specification introduces the services provided by SMDP2.0, and specifies its service interfaces, protocols and coding, as well as steps to obtain market data from the Platform.

2. Market Data and Its Organization Structure

2.1 Topic

SHFE's market data on futures and derivatives trading is organized based on topics. Each topic has defined what market data is released and how it is released, including market depth, sample frequency, delay time, statistical model, etc. For the detailed parameters of topics, refer to the list of market data topics released by SHFE.

- **Product:** It refers to the varieties of futures and derivatives covered by the topic.
- **Market Depth:** SMDP2.0 shows Market By Price (MBP), so market depth refers to the number of price levels.
- **Sample Frequency:** It refers to the frequency of snapshot.
- **Delay Time:** It refers to the difference between a market data snapshot and its real-time snapshot. The delay time of a real-time market data snapshot is 0.
- **Statistical Model:** It is divided into bilateral statistics and unilateral statistics. In unilateral statistics, trade volume and value are calculated unilaterally; while in bilateral statistics, they are calculated on a bilateral basis, so the trade volume and value are twice that in the unilateral statistics.

The users of the Platform have access to the market data of different topics corresponding to their permissions.

2.2 Market Data Snapshot and Market Data Incremental

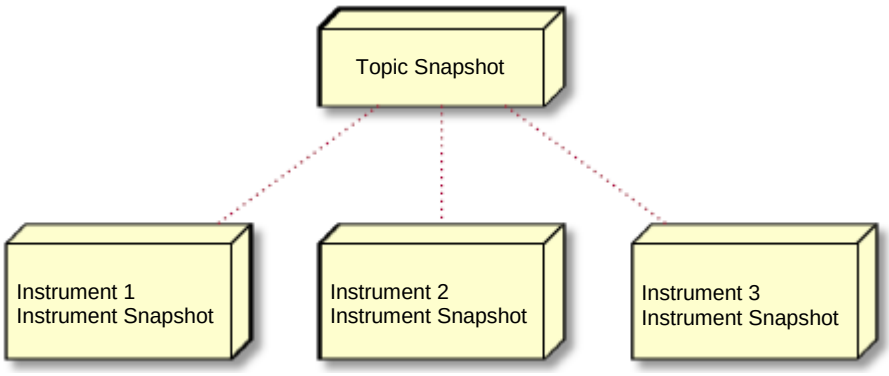
The Platform provides market data to customers via snapshots and incremental.

- **Market Data Snapshot** ("Snapshot" in short) reflects the market status at a certain snapshot time (the moment when a snapshot is made), including all market information such as orders, trades, instruments and market statistics at that moment.
- **Market Data Incremental** ("Incremental" in short) shows the market changes in snapshots, including all the changes between the corresponding two snapshot times.

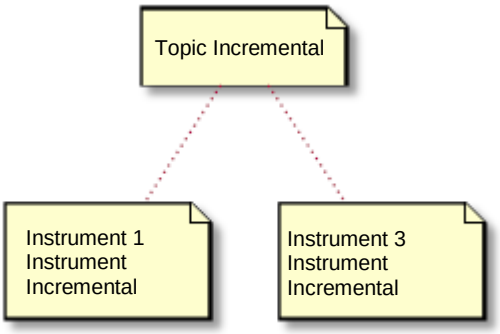
Such changes to market data lead to the update of snapshots, which in turn generates the corresponding incremental. One new snapshot can be obtained according to its pre-snapshot and corresponding incremental.

One instrument snapshot is the snapshot of an instrument at a specific snapshot time. One topic snapshot of a topic is composed of instrument snapshots at the same snapshot time of all futures and options contracts included in the topic. In case an instrument does not generate incremental (no updates) at a snapshot time, the topic incremental does not include instrument snapshot of this instrument.

> For example: Suppose a topic includes three instruments: Instrument 1, Instrument 2 and Instrument 3, then a topic snapshot is composed of the instrument snapshot of Instrument 1, the instrument snapshot of Instrument 2, and the instrument snapshot of Instrument 3.

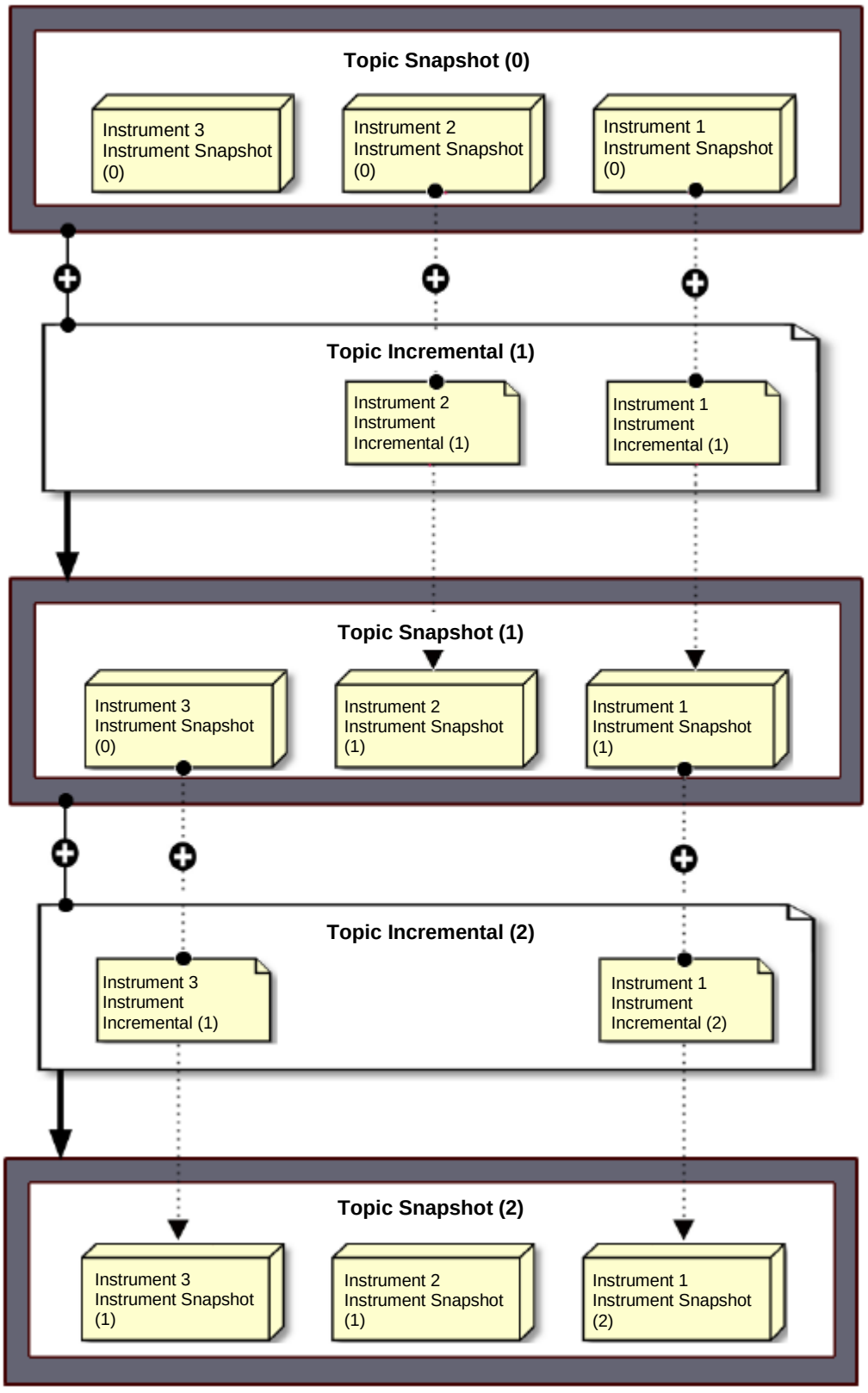


> If at a certain snapshot time, only the instrument snapshots of Instrument 1 and Instrument 3 are changed, then the topic incremental includes instrument incremental of Instrument 1 and Instrument 3 only. Instrument 2's blank instrument incremental is omitted.



Each topic independently numbers its topic snapshots, called snapshot number. Snapshot number starts from 0 and increases by one each time. Topic Incremental is numbered as well. When the snapshot number of a topic snapshot changes from n to n+1, then the incremental number of the corresponding topic incremental is n+1. After obtaining a topic snapshot, a user can update subsequent topic snapshots based on the subsequent topic incrementals. The incremental number of each topic incremental is the same as the snapshot number of the corresponding topic snapshot. In following chapters, incremental number is sometimes also referred to as snapshot number.

Instrument snapshot is numbered as well. Except that the initial number is the same as the snapshot number of the topic snapshot, subsequently it is independently numbered according to changes to the instrument. An instrument incremental is connected with its instrument snapshot by number as well. Each instrument incremental leads to the increase of the snapshot number of its corresponding instrument snapshot. The incremental number of each instrument incremental is the same as the snapshot number of the corresponding instrument snapshot. Please note that every time the topic snapshot changes, the snapshot number must be increased accordingly. However, the topic incremental does not include the instrument incremental of all instruments. If only some instrument snapshots are increased in snapshot number due to a change to the topic incremental, then the incremental number of the topic incremental may gradually become different from the incremental number of some instruments under the topic.



Each instrument snapshot is composed of three parts: instrument information, trade quotation and MBP (Market by Price) list. While instrument incremental is composed of a series of instrument events, and each instrument event represents the change of a specific part of the information in the snapshot. Details are provided in subsequent chapters.

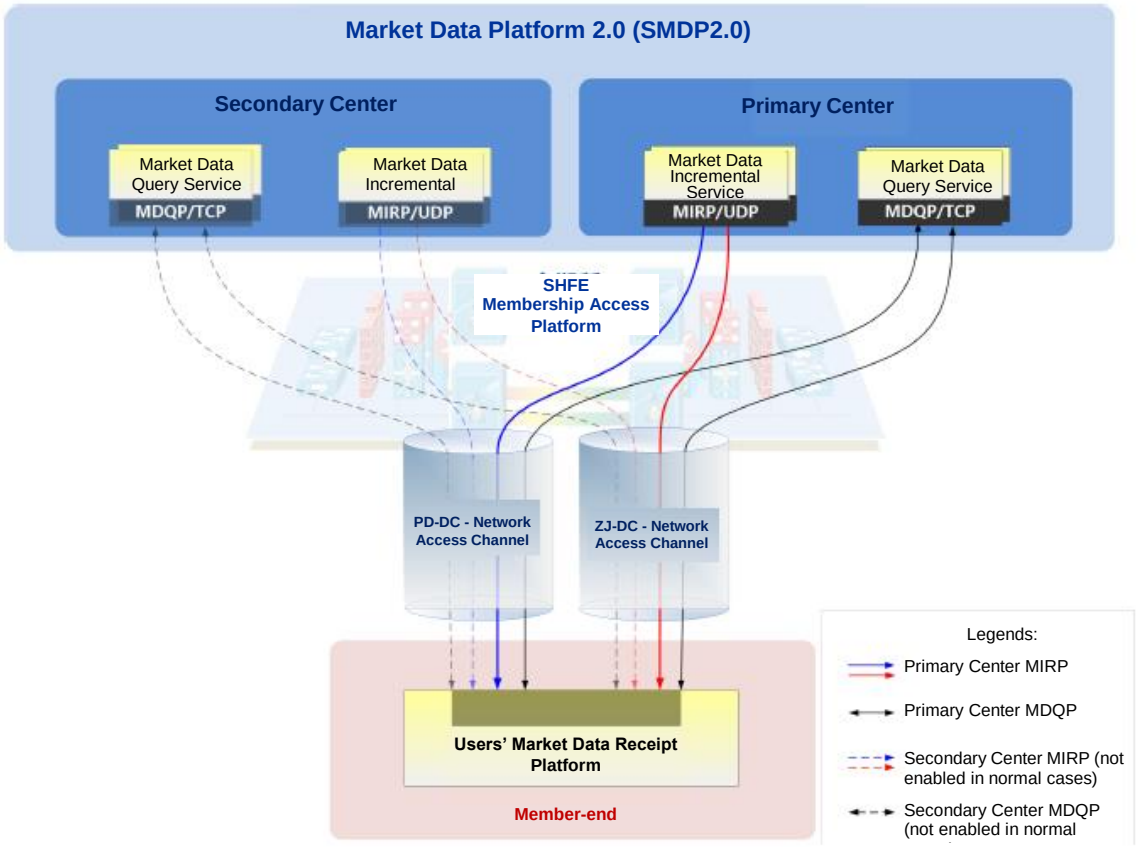
3. Services

SMDP2.0 snapshots the market data of the products at the sample frequency defined by each topic. In case of changes to market data, a corresponding topic snapshot is generated, and the resulting topic incremental will be sent via multicast, so as to provide users with market updates.

The Platform provides market data query service in interaction with users and market data incremental service that is unidirectionally transmitted to users.

- **Market Data Query Service** allows users to log in/ out, query market data snapshots and supplement missing market data incrementals. This function applies MDQP (Market Data Query Protocol) to interact with users. MDQP's network layer protocol is TCP.
- **Market Data Incremental Service** sends the topic incrementals to their subscribers. This function applies MIRP (Market data Incremental Refresh Protocol) to inform users in a multicast manner. MIRP's network layer protocol is UDP.

SMDP2.0 is deployed in three data centers located in Shanghai and Beijing, and runs under primary-secondary parallel mode. Each center is equipped to provide several market data query services and market data incremental services. For market data query services, users may choose any of them. For market data incremental services, users can connect to one or more lines at the same time. Under normal circumstances, Secondary Center is not enabled (shown by dotted lines). See the figure shown below.



4. Member Type and Packet

Prior to the introduction of the Platform's functions and protocols, this chapter offers a summary of the member types and packet structures supported by the two protocols.

4.1 Member Type

Member is the basic unit of information modules in the two protocols used by the Platform. Each member is encoded according to its type. Members are encoded into larger information modules in a little-endian and compact manner in an order defined by the protocols. Byte alignment is not supported.

The encoding types adopted by the Platform include: integers of varied sizes, double-precision floating point, character arrays and byte arrays of different lengths, etc. The details are shown in the table below.

Type	Length (byte)	Encoding	Description
Int*n*	*n*/8	Binary	A signed integer of length *n* bits; *n* can be 8, 16, 32 or 64. For example, **Int**32 is a 32-bit binary signed integer.
uInt*n*	*n/8*	Binary	An unsigned integer of length *n* bits; *n* can be 8, 16, 32 or 64.
Double	8	Binary (IEEE754)	Double precision floating-point; DBL_MAX is used to indicate invalid values.
Char[*n*]	*n*	Binary	A standard character or string of length *n*; *n* can be any positive integer. When *n* is 1, it means character; when *n* is greater than 1, it is a character string. A string must end with "\0". In other words, the valid character string stored in Char[*n*] (*n*> 1) is the character sequence before the first "\0" stored in it.
Byte [*n*]	*n*	Binary	A byte array of length *n*; *n* can be any positive integer.
Vint	1∽10	Varint+ZigZag	A compact method for encoding 64-bit signed integers.

Vint is a combination of Varint and ZigZag. It encodes 64-bit binary coded signed integers with a maximum of 10 bytes.

- Varint is a method that encodes 64-bit binary unsigned integers with bytes of varied length according to their size. Its feature is that the smaller the number, the fewer the number of bytes occupied. The lower 7 bits of each byte are used to represent the number, and the highest bit is given a special meaning: if the bit is 1, it means that the subsequent byte is also part of the number; if the bit is 0, the byte is the last. Therefore, all numbers less than 128 can be represented by one byte; and those greater than 128 are encoded with two or more bytes.

  > For example:
  > Integer 1 is represented by one byte only
  > 0000 0001
  >
  > Integer 300 needs to be represented by two bytes
  > 1010 1100 0000 0010

- **ZigZag** is used to define signed integers for use with Varint. It re-parses the binary according to the absolute value of integers. The principle is shown in the following table:

Number of original signs	Encoded as
0	0
-1	1
1	2
-2	3
2	4
-3	5
…	…
2147483647	4294967294
-2147483648	4294967295

Its encoding expression is:

EncZZ(n)= (n <<1) ^ (n >>63)
Wherein: << refers to bitwise left shift operation, ^ a bitwise XOR operation, >> bitwise right shift operation.

Its decoding formula is:

DecZZ(n)= (-(n & 0 x 01)) ^ ((n >>1) & ~(1<<64))
Wherein: & refers to bitwise AND operation, ~ bitwise NOT operation.

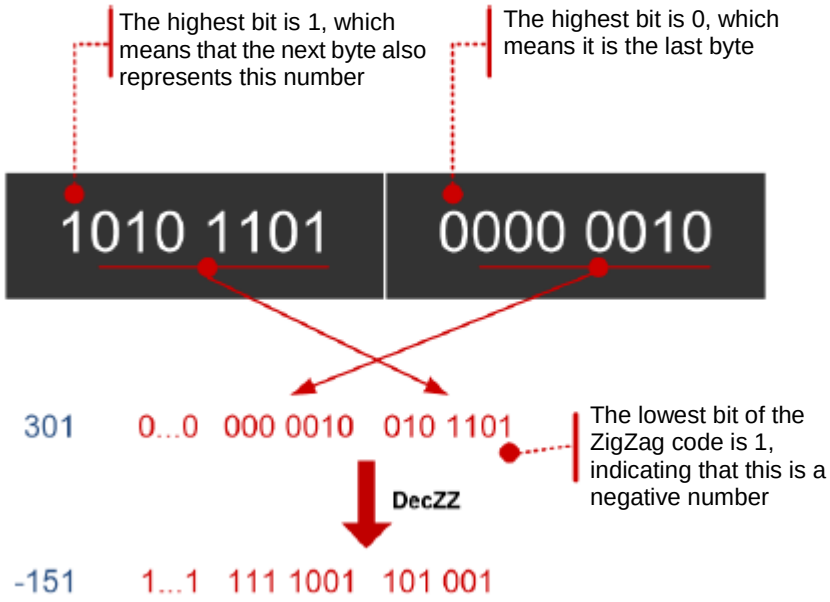
Vint is to first convert a 64-bit signed integer with ZigZag, and then encode it with Varint. Vint encoding is the same as that of sint64 in wire type in Google Protocol Buffer.

The algorithm for converting **Vint** codes to 64-bit signed integers is divided into two steps:

- Step 1: First read the target as a Varint code and decode it into a 64-bit unsigned integer with ZigZag.
  1. Clear all 64 bits of the ZigZag code to 0;
  2. Starting from the first byte, sequentially take out one byte of the Varint code, assumed as byte i;
  3. The lower 7 bits of the retrieved byte are cumulatively assigned to the ZigZag code: the bit n of the Varint coded byte (n is 1~7) is assigned to the bit (i-1)*7+n of the ZiaZag code;
  4. Judge the highest bit of the retrieved byte: if it is 1, then return to Step 1 to take the next byte for decoding; if it is 0, the Step 1 ends.

- Step 2: Convert the resulting 64-bit unsigned integer encoded by ZigZag to a 64-bit signed integer with the DecZZ formula.

The following figure shows the decoding process of -151:



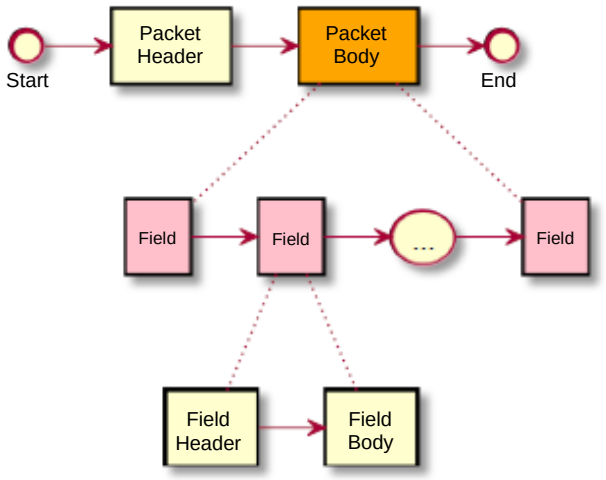
4.2 Packet

Both MDQP and MIRP are application layer protocols, which are the basis for data interaction between the Platform and the client end. Each complete information transfer between the Platform and the client end is called a message. Messages are transmitted in the form of packets.

Packets sent via MDQP are called MDQP packets, and those sent via MIRP are MIRP packets.

4.2.1 Packet Structure

Each packet consists of a packet header followed by a packet body.



The packet header is a sequence of members. The composition of members in a packet header varies from protocols, yet generally including information such as message type, packet ID, and body length. The packet body is composed of one or more fields. Fields vary from field types. A field ID uniquely identifies a field type in a protocol.

A field is composed of a field header and a field body. Both field header and field body are member sequences. The field header includes two members - field ID and body size. For different types of fields, the members in the field bodies are defined variedly. The structure of a field header is as follows:

Member	Type	Description
FieldID	Int16	Field ID; uniquely represents a field type.
FieldSize	Int16	Lengthen of the field body.

Each message is composed of one or more fields. The decoding procedure of a protocol packet depends on field headers to divide the packet body into different fields. The packet body length indicated in the header (this member is generally present in packets) equals the sum of the lengths of all field bodies (“FieldSize” in the header) plus the length of all field headers.

Note: The field body length “FieldSize” indicated in the field header may be greater than the sum of the member lengths in the field definition. This case usually occurs at the time of version updating, which adds new field members of field body in the end, while the client end is still using the old version for protocol decoding as it is forward compatible. Therefore, when dividing a packet body into multiple fields, please be sure to use the field body length in the field header, instead of the field body member definition. Redundant bytes at the end of the field body, if any, should be discarded, instead of used as the beginning member of the next field, otherwise the packet decoding will go wrong.

4.2.2 Field Composition Grammar

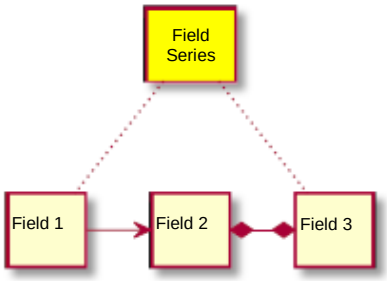
As mentioned in the previous section, a message is composed of one or more fields, and the types and sequences of fields vary from message types. In this section the field composition grammar is defined. Subsequent paragraphs will follow this grammar to describe the field composition of messages.
Two sets of grammar - regular expression and structure diagram, are defined to represent the field composition of a message. Users can choose the one they are familiar with.

- Parentheses and names are used to indicate a field or field string, for example:
  (field)

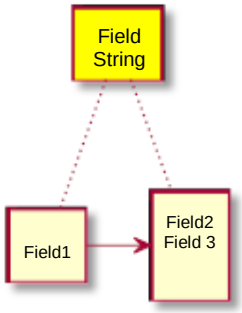
  represents a field. At the same time, parentheses may appear in regular expressions to help indicate priority.
  In structure diagrams, rectangles are used to represent fields, and rectangles in yellow represent field series.

- Successive arrangement in a regular expression represents fields in sequence. When two fields are put in the same pair of parentheses, it means there is no requirement for sequence (the order can be changed). For example:
  (Field 1) (Field 2, Field 3)

  It indicates that this field series must be in the order of Field 1 followed by Field 2 and Field 3.
  In a structure diagram, two dotted lines are used to represent the specific components of a field string. The order is represented by arrows between fields and lines with diamonds on two ends mean there is no requirement on sequence.



More than one member fields with no order requirements can also be placed in the same rectangle.



- Superscripts are used to indicate the multiple appearances of a field or field series. It is a limit on the number of appearances of a field or field series. For example:
  (Field)$^{>=1 \wedge <=M}$

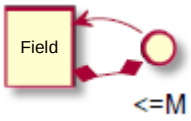
It indicates that the field appears at least once, and at most M times. "/\" is used to represent “AND" as a restriction condition, and "V" as “OR". If the superscript is a fixed number, the number itself serves as a restriction. For example, "1" means that the field will appear once, so the following two regular expressions are equivalent.

(Field) equals $(Field)^{1}$

The following expression indicates that the field can appear any times.

$(Field)^{>=0}$

In a structure diagram, a circle with an arrow is used to represent the restricted number of appearances of a field or field series. Below the circle shows the limited number of appearances.



- Fields in the same pair of parentheses are subject to the same restriction. For example:

  $(Field\ 1, Field\ 2)^{>=1/\backslash<=2}$

  Possible field series are "Field 1 Field 2", "Field 2 Field 1", "Field 1 Field 1 Field 2", "Field 2 Field 1 Field 1", "Field 2 Field 2 Field 1", and "Field 1 Field 2 Field 2". Please note that these fields must appear together, and no other fields can be inserted in between.
  The same rule applies to the fields in the same rectangle.

4.2.3 Message and Packet

For both protocols, packets are subject to a maximum length:

- MDQP: 1,280 bytes;
- MIRP: 1,232 bytes.

These upper limits include packet headers as well. So, the sum of the header length of a MDQP packet and the body length in the packet header shall be no more than 1,280 bytes; the sum of the header length of a MIRP packet and the body length in the packet header shall be no more than 1,232 bytes.

When there are too many fields to be packed in one packet for one message, it is necessary to segment the data. At the time of segmenting, please ensure the integrity of fields, that is, do not segment the same field into two parts and put them into different packets. For some special messages, there may be other special segmentation requirements.

A message may be composed of more than one packets, but a packet can only contain the data of one message at most. Different messages should not share the same packet. Even if the remaining packet length is sufficient, a new message still needs to be encapsulated by another packet. Sometimes a message type is also referred to as packet type.

5. Market Data Query Service

This chapter introduces the MDQP message types of market data query service and their corresponding field composition.

5.1 MDQP Packet Header

The MDQP packet header contains information such as the protocol version number, message type, packet body length, and user request number. The total length of the MDQP packet header is 8 bytes.

Member	Type	Description
Flag	ulnt8	Flag bits and protocol version number. The higher 4 bits are 4 flag bits; the lower 4 bits (Flag & 0x0F) are the protocol version number, and the current version number is 1. The lowest bit among the higher 4 bits (i.e. the 4th bit, Flag & 0x10) is the message end flag bit. When the ID bit is 0, it means that the packet is the last packet of the message; when it is 1, it means that the subsequent packet still belongs to the same message. The remaining 3 flag bits have not been used.
TypeID	Int8	Message type, or packet type, represents a type of message uniquely.
Length	ulnt16	Length of packet body.
RequestID	Int32	Number of user request. In MDQP, when a user makes a query request, he/ she needs to provide a request number to uniquely identify the request. The Platform brings back the request number in the corresponding response packets to maintain the correspondence between the request and the response packets.

5.2 Functions

Market Data Query Service provides users with functions including heartbeat, log in/ out, inquire about topic snapshot query and topic incremental query.

5.2.1 Heartbeat

Upon connection between a user and the Platform’s market data query service, when no packet is received from one side by the other within a specified time, the connection times out. The default timeout value is 10 seconds.

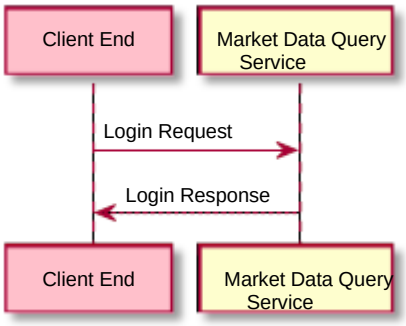
The heartbeat function is used to notify the other side that the connection is valid. If, within a certain period of time, no packet is required to be sent, one side needs to send a heartbeat message to the other side to avoid connection timeout. In case of connection timeout, it means the connection is invalid and the network should be disconnected.

Heartbeat Message (TypeID=0x00)

Each heartbeat message consists of a blank packet with only a MDQP packet header. The body of the blank packet should be blank, and does not contain any fields. In the heartbeat message header, “Length” is 0, and “RequestID” member is left void.

5.2.2 User Login

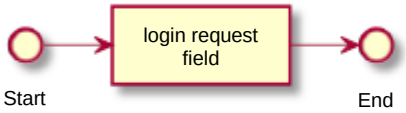
Before starting to receive market data, a user must first log into the Platform. The user first needs to send a login request message to the Platform's market data query function, then the Platform sends a login response message for feedback.



Login Request (TypeID=0x11)

A login request message needs one login request field only.

(login request field)



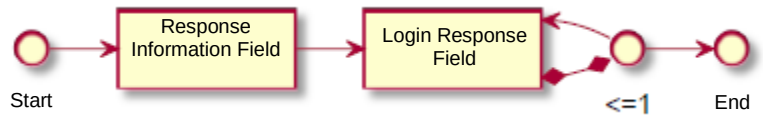
- **The login request field** (FieldID=0x0002) includes the information provided when the user applies for login.

Member	Type	Description
UserID	Char[16]	User's code.
ParticipantID	Char[11]	Participant's code.
Password	Char[41]	Password.
Language	Char[1]	Language; the decoding language affecting members such as "ErrorMsg" member. Its value range is as follows: '0': Chinese (GB18030) '1': English
UserProductInfo	Char[41]	Information on user's product; used to indicate which market data system is used by the client.
InterfaceProductInfo	Char[41]	Information on interface product. As protocol decoding software and market data software used by users may be provided by different manufacturers, it is used to indicate which protocol decoding software the user uses.

Login Response (TypeID=0x12)

If the login is successful, the login response message includes the response information field and the login response field. If the login fails, the login response message contains the response information field only.

(response information field) (login response field)$^{<=1}$



- **The response information field** (FieldID=0x0001) contains error code and error message, notifying the user whether the login is successful. In market data query service, the response information field is also the only field in the feedback message following any query failure.

Member	Type	Description
ErrorID	Int32	Error code, represents a type of error uniquely. 0 indicates no error.
ErrorMsg	Char[81]	Detailed information on the error.

The Platform has defined all the error codes as follows:

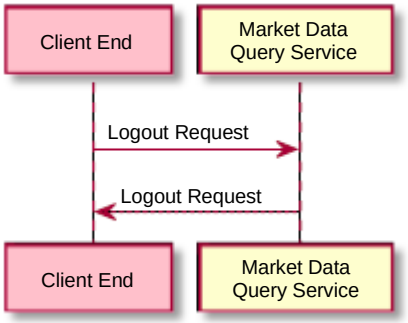
Error Code	Reason
0	No Error
1	Too High MDQP Version
3	Participant Not Exist
-22	Operation Too Frequent.
-4118	Exchange Data Not Ready
-4155	Duplicate Login
-4156	Invalid User Or Password.
-4158	User Inactive
-4160	User Not Belong To This Participant
-4161	Invalid User IP.
-4162	User Not Login.
-4163	Not Logined By This User
-4164	Not Logined By This Participant
-4202	Duplicate Session
-4203	No Function Right
-4204	Error Request Packet Number.

- **The login response field** (FieldID=0x0003) only appears when the login is successful (error code in the response information field=0); the information and members fed back to the user are composed as follows:

Member	Type	Description
TradingDay	Char[9]	The day of trading; the format is "YYYYMMDD\0".
LoginTime	Char[9]	The time of successful login; the format is "hh:mm:ss\0”.
UserID	Char[16]	User’s code.
ParticipantID	Char[11]	Participant’s code.
TradingSystemName	Char[61]	The name of the system.
ActionDay	Char[9]	The calendar date; the format is the same as that of the trading day.

5.2.3 User Logout

To log out, the user needs to send a logout request message to the Platform, then the Platform sends a logout response message for feedback.



Logout Request (TypeID=0x13)

A logout request message needs one logout request field only.

(logout request field)

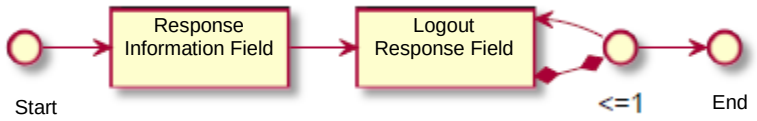
- **The logout request field** (FieldID=0x0004) includes the information provided when the user applies for logout.

Member	Type	Description
UserID	Char[16]	User's code.
ParticipantID	Char[11]	Participant's code.

Logout Response (TypeID=0x14)

If the logout is successful, the logout response message includes the response information field and the logout response field. If the logout fails, the logout response message contains the response information field only.

(response information field)(logout response field)$^{<=1}$

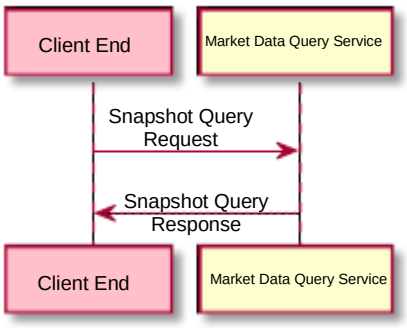


- **The response information field is the same as that in login response.**
- **The logout response field** (FieldID=0x0005) is the final feedback made by the Platform to the user after he/ she successfully logs out.

Member	Type	Description
UserID	Char[16]	User's code.
ParticipantID	Char[11]	Participant's code.

5.2.4 Topic Snapshot Query

A user sends a request for topic snapshot query to obtain the full market data at a specified snapshot time, which is used as initial full market data or market data recovery. After receiving the request, the Platform's Market Data Query Service sends a snapshot query response message to the user's query.

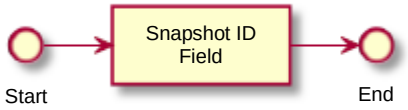


Note: Users can only query topic snapshots, and are not allowed to query instrument snapshots.

Snapshot Query Request (TypeID=0x31)

A snapshot query request message only needs a snapshot ID field to uniquely identify a topic snapshot.

> (snapshot ID field)



- **The snapshot ID field** (FieldID=0x1001) contains the topic code and snapshot number. When snapshot number is -1, it indicates that the latest snapshot needs to be queried.

Member	Type	Description
TopicID	Int16	Topic's code.
SnapNo	Int32	Snapshot number. -1 represents the latest snapshot.

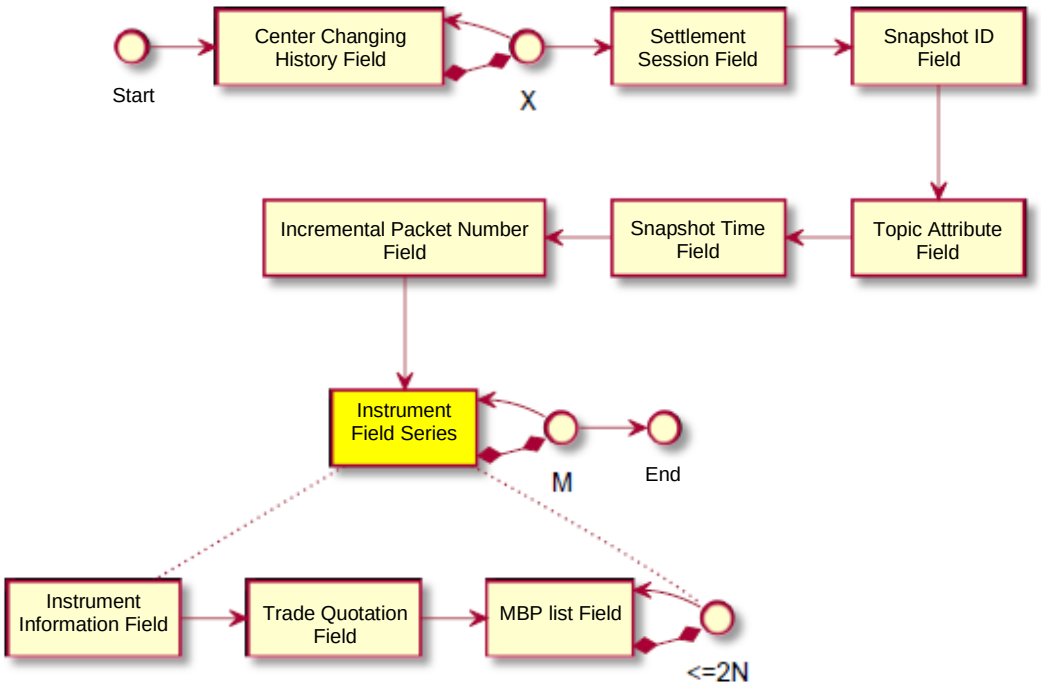
Snapshot Query Response (TypeID=0x32)

The field composition of a snapshot query response message is as follows:

- X center changing history field (s) (X refers to the number of changes of data center on a trading day, and 0 indicates there is no change on the day.)
- A settlement session field
- A snapshot ID field
- A topic attribute field
- A snapshot time field
- A incremental packet number field
- M instrument field series (M is the number of instruments in the topic). The composition of each instrument field series is:
  - A instrument information field
  - A trade quotation field
  - <=2N MBP list field(s) (N is the market depth of the topic)

In regular expression and structure diagram, the field composition of a snapshot query response message is shown respectively as follows:

> ((center changing history field) ^{X}) (settlement session field) (snapshot ID field) (topic attribute field) (snapshot time field) (incremental packet number field) (((instrument information field) (trade quotation field) ((MBP list field) $^{<=2N}$)) ^{M})



- **The center changing history field** (FieldID=0x0032) comprises the data center's changing history after the start of the trade system, and is used to provide information on the data center's changing history only. This field mainly includes the valid snapshot number for each center switch and its corresponding incremental packet number. Information contained in the center changing history member is generally provided to users for effective market performance judgment in the event of data center change.

Member	Type	Description
CenterChangeNo	Int8	Data center number. The data center number begins at 0 each time the system starts, and increases by 1 each time data center changes.
SnapNo	Int32	Valid snapshot number at the time of the current data center change.
PacketNo	Int32	Incremental packet number of the current data center change. The changed data center will send incremental packets starting with the packet number PacketNo + 1.

- **The settlement session field** (FieldID=0x0031) comprises information related to settlement.

Member	Type	Description
TradingDay	Char[9]	The day of trading; the format is "YYYYMMDD\0".
SettlementGroupID	Char[9]	Settlement group's code.
SettlementID	Int32	Settlement's code.

- **The snapshot ID field** is the same as that in snapshot query request.
- **The topic attribute field** (FieldID=0x1003) includes topic-related information such as market depth and encryption of incremental MIRP packets.

Member	Type	Description
MarketDataDepth	Int32	Market data depth of the topic.
CipherAlgorithm	Char[1]	Encryption algorithm. Encryption is currently not enabled. Its value range is: '0': No encryption.

Member	Type	Description
CipherKey	Byte[16]	Key, only valid when the encryption algorithm is not '0'.
CipherIV	Byte[16]	Encryption’s initialization vector, only valid when the encryption algorithm is not '0'.

- **The snapshot time field** (FieldID=0x1002) comprises the information on market data of a topic at a snapshot time.

Member	Type	Description
SnapDate	Char[9]	The date of snapshot. the format is "YYYYMMDD\0".
SnapTime	Char[9]	The time of snapshot (precise to second). the format is "hh:mm:ss\0”.
SnapMillisec	Int32	The time of snapshot (millisecond-level).

- **The incremental packet number field** (FieldID=0x1004) is used to indicate the latest number of incremental packet in the current snapshot of the topic. This means that the current snapshot reflects the aggregate information of all incremental messages marked with a number that is lower than or equal to this number.

Member	Type	Description
PacketNo	Int32	The latest number of incremental packet.

- **The instrument information field** (FieldID=0x0101) is used to encode the instrument information in an instrument snapshot. Instrument information includes the attributes and status of an instrument, and generally remains unchanged within a trading day.

Member	Type	Description
InstrumentID	Char[31]	Instrument code; the unique ID string of an instrument. SHFE and related systems use instrument codes to refer to corresponding instruments.
UnderlyingInstrID	Char[31]	Underlying instrument code.
ProductClass	Char[1]	The product class to which the instrument belongs, that is, the product type. All instruments under a product are of the same type. Product classes are divided into futures, options, combination, spot, EFP, TAS, spread and indexes, valued as follows: ‘1’: Futures ‘2’: Options ‘3’: Combination ‘4’: Spot ‘5’: EFP ‘6’: TAS ‘7’: Spread ‘I’: Indexes.
StrikePrice	Double	Strike price, exclusive to options instruments, indicates the strike price of an options instrument.
OptionsType	Char[1]	The type of the options, exclusive to options instruments, divided into bullish (call) and bearish (put). It is valued as follows: ‘0’: Non-option ‘1’: Bullish (call) ‘2’: Bearish (put)

Member	Type	Description
VolumeMultiple	Int32	The multiplier of instrument volume, indicates how many units of the subject matter the instrument contains per lot.
UnderlyingMultiple	Double	The multiplier of the underlying instrument.
IsTrading	Int32	It indicates whether it's in trading state. 0 means non-trading state, and others refer to trading state.
CurrencyID	Char[4]	The currency code of the instrument, valued as "CNY" or null: "CNY" means RMB; null value is used for indexes.
PriceTick	Double	Tick size. Instrument price must be the integral multiple of the instrument's tick size.
CodecPrice	Double	Base price for encoding; special information, specified by the Platform. It is generally the settlement price of the previous day, and used as a codec benchmark for some price members in market data incremental.
InstrumentNo	Int32	Instrument code, corresponds to an instrument uniquely. The instrument code is mapped to 32-bit integer encoding to reduce the number of bytes.

- **The trades quotation field** (FieldID=0x0102) is used to encode all basic market data information in an instrument quotation snapshot, indicating the price, statistical information and some status information of the instrument. The statistical information of instruments is based on a statistical time period. The statistical information provided by the Platform is compiled by trading day; that is, the information is recorded and calculated from the opening of the trading day to the current time.

Member	Type	Description
InstrumentNo	Int32	Instrument code, corresponds to the instrument code in the instrument information field.
LastPrice	Double	The latest price.
Volume	Int32	Instrument volume; the number of lots that have been traded on the day of trading.
Turnover	Double	Turnover; the total transaction value of the instrument on the trading day.
OpenInterest	Double	Open interest; the total open interest of the instrument in the market.
HighestPrice	Double	The highest price; the highest trading price so far on the trading day.
LowestPrice	Double	The lowest price; the lowest trading price so far on the trading day.
OpenPrice	Double	Today's open price.
ClosePrice	Double	Today's close price.
SettlementPrice	Double	Today's settlement price.
UpperLimitPrice	Double	Upward limit price.
LowerLimitPrice	Double	Downward limit price.
PreSettlementPrice	Double	Yesterday's settlement price.
PreClosePrice	Double	Yesterday's close price.
PreOpenInterest	Double	Yesterday's open interest; the total open interest of the instrument before the opening on the trading day.
PreDelta	Double	Yesterday's delta value.

Member	Type	Description
CurrDelta	Double	Today's delta value.
ActionDay	Char[9]	The calendar date.
UpdateTime	Char[9]	The last update time (second)
UpdateMilliSec	Int32	The last update time (millisecond)
ChangeNo	Int32	The snapshot number of the current instrument snapshot.

- **The MBP list field** (FieldID=0x0103) includes all the MBP list information in an instrument quotation snapshot. A MBP list is a subset of the information in the order book, and indicates the distribution of the order on a specified price level for the day. The Platform provides MBP list information in the sequence of price ranking. The price level depth in the price table information is the market depth of the topic market data. Assuming the market depth is N, then the MBP list of an instrument incorporates the information of 2N price levels. There are N levels in the bid direction, and the best level locates the highest price; there are N levels in the ask direction, and the best level has the lowest price.
  Each MBP list field represents the price information of a level in one direction, so the MBP list information with market depth of N needs at most 2N MBP list fields for encoding.

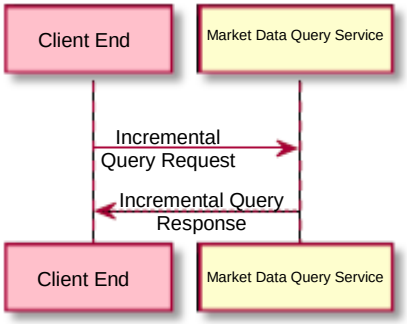
Member	Type	Description
InstrumentNo	Int32	Instrument code, corresponds to the instrument code in the instrument information field.
Direction	Char[1]	Bid-ask direction. Its value range is as follows: ‘0’: Bid ‘1’: Ask.
Price	Double	Price level.
Volume	Int32	Instrument volume;

Note: For the definitions of all related fields in market data snapshots, please refer to detailed rules of the Exchange.

5.2.5 Topic Incremental Query

In case a small number of topic incremental packets are lost, they may be supplemented via the topic incremental query function. The Platform has limited that a user may query no more than 10 messages at a time.

To supplement such packets, the user has to send an incremental query request to the Platform, and the Platform then sends an incremental query response message as feedback.



Incremental Query Request (TypeID=0x33)

A incremental query request message contains only one incremental packet ID field, which is used to identify a continuous series of incremental packets.

(incremental packet ID field)

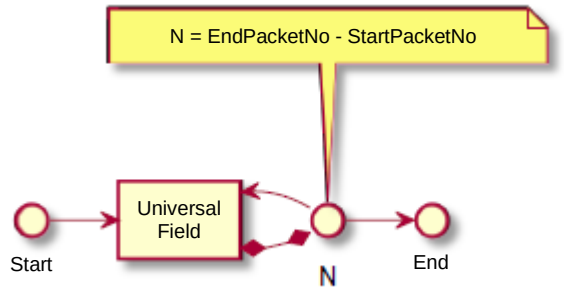
- **The incremental packet ID field** (FieldID=0x0201), besides a topic code, also contains two packet numbers: StartPacketNo and EndPacketNo, indicating that the number range of the incremental packet to be queried is [StartPacketNo, EndPacketNo), and the sum of packets to be queried is EndPacketNo-StartPacketNo.

Member	Type	Description
TopicID	Int16	Topic's code.
StartPacketNo	Int32	The starting packet number (included).
EndPacketNo	Int32	The ending packet number (excluded).

Incremental Query Response (TypeID=0x34)

An incremental query response message transmits incremental packets in the form of universal fields, and each universal field contains one incremental packet.

(universal field)$^{(EndPacketNo - StartPacketNo)}$



- **The universal field** (FieldID=0x0000) consists of a body with undefined binary data. It is universal and can be used to send any data. However, MDQP cannot decode a universal field body, and generally requires an additional algorithm. For example, topic incremental packets are decoded via MIRP.

6. Market Data Incremental Service

This chapter introduces the MIRP message types of market data incremental service and their corresponding field composition.

6.1 MIRP Packet Header

The members contained in a MIRP packet header are listed as follows from front to back:

Member	Type	Description
Flag	uInt8	ID bits and protocol version number. The higher 4 bits are 4 ID bits; the lower 4 bits (Flag&0x0F) are the protocol version number, and the current version number is 1. The lowest bit among the higher 4 bits (i.e. the 4th bit, Flag&0x10) is the message end ID bit. When the ID bit is 0, it means that the message is the last message of the packet; when it is 1, it means that the subsequent message still belongs to the same packet. The remaining 3 ID bits have not been used.
TypeID	Int8	Message type, or packet type, represents a type of message uniquely.
Length	uInt16	Length of packet body (excluding packet header).
PacketNo	Int32	Market data packet number, uniquely marks a incremental packet. Heartbeat packets inherit the packet number of the latest incremental packet.
TopicID	Int16	The topic code of the market data included in the packet body.
SnapMillisec	uInt16	The millisecond-level snapshot time of the current topic incremental. Heartbeat packets inherit the millisecond-level time of the latest incremental packet.
SnapNo	Int32	The snapshot number of the current topic incremental. Heartbeat packets inherit the snapshot number of the latest incremental packet.
SnapTime	uInt32	The second-level snapshot time of the current topic incremental. Heartbeat packets inherit the second-level time of the latest incremental packet.
CommPhaseNo	uInt16	The number of days from January 01, 1980, to represent the current trading day.
CenterChangeNo	Int8	Data center number, used in case of data center change. The data center number begins at 0 each time the system starts, and increases by 1 each time data center changes.
Reserved	Int8	Reserved bytes.

The length of a MIRP packet header is 24 bytes.

If the MIRP packet is encrypted, it means only the body of the MIRP packet is encrypted. When receiving the MIRP packet, you can analyze the packet header to selectively decrypt the packet.

6.2 Functions

This section introduces the functions of market data incremental service and corresponding message composition.

6.2.1 Heartbeat

The heartbeat function is enabled by heartbeat messages. The Platform sends a heartbeat message every 3 seconds.

Heartbeat Message (TypeID=0x00)

Each heartbeat message consists of one heartbeat packet. A heartbeat packet is a blank MIRP packet. Heartbeat messages are used to assist users to judge the current market data incremental service status and market data status. As they are not classified as incremental packets, heartbeat packets do not trigger the increase of incremental packet number. The packet number and snapshot number in the header of a heartbeat packet can help users judge the current market status.

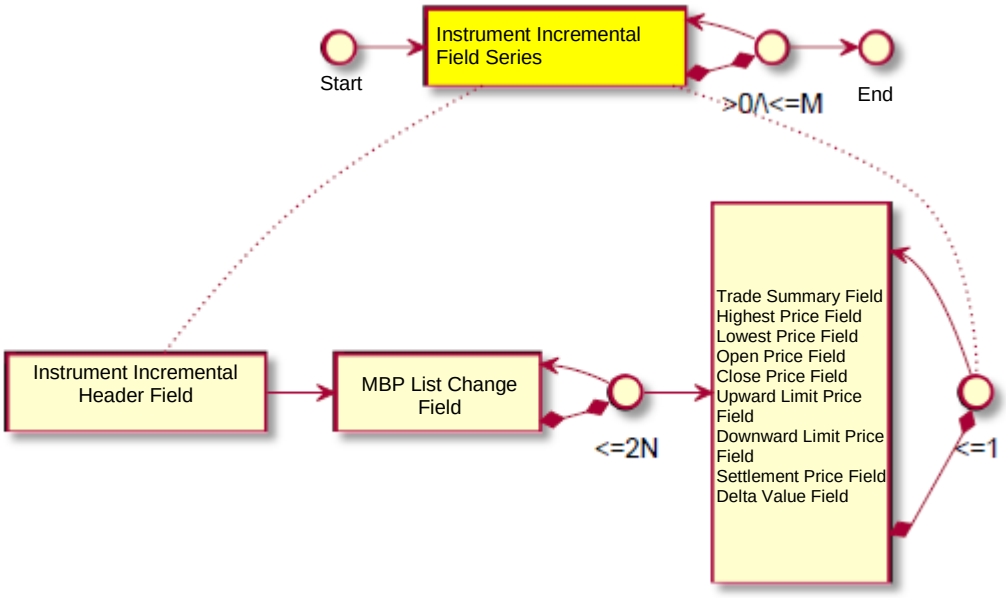
6.2.2 Incremental Feed

The Platform offers the incremental feed function, which broadcasts topic incrementals to all users in a broadcast group at the refresh rate of the corresponding topic.

Incremental Refresh Message (TypeID=0x01)

An incremental refresh message is composed of one to M (the number of instruments included in the topic) instrument incremental field series. In case an incremental refresh message is too large and needs to be divided into multiple packets, the instrument incremental field series of the same instrument must be in the same packet. Each instrument incremental field series starts with a field header, followed by 0 to 2N (N refers to the market depth) MBP list change fields, and then a series of other instrument event related fields. Except for MBP list change fields, other event fields appear at most once.

> ((instrument incremental field header)(MBP list change field)$^{<=2N}$(trade summary field, highest price field, lowest price field, open price field, close price field, upward limit price field, downward limit price field, settlement price field, delta value field)$^{<=1}$)$^{>0/\<=M}$



- **The instrument incremental header field** (FieldID=0x0003) contains the instrument code and instrument incremental number, which is generally used to identify the beginning of an instrument incremental.

Member	Type	Description
InstrumentNo	Vint	Instrument code,
ChangeNo	Vint	The number of the current instrument incremental; when the corresponding instrument snapshot changes, the number increases by one.

- **The MBP list change field** (FieldID=0x1001) contains the data of a MBP list change event. A MBP list change event indicates a change to the MBP list in the instrument snapshot, i.e. the change of information on a specified price level in one direction of the MBP list. One single MBP list change event may not completely show all MBP list changes, so an instrument incremental may include more than one MBP list change events.

Field	Type	Remarks
EventType	Char[1]	The type of the event. Its value range is: '1': Add '2': Change '3': Delete.
MDEntryType	Char[1]	Bid-ask direction. Its value range is: '0': Bid '1': Ask.
PriceLevel	Vint	Number of price levels. Increasing from 1 in order, it indicates the level of the MBP list to be operated.
PriceOffset	VInt	Level price. This value refers to the deviation of the tick size between the actual price and the codec reference price in the snapshot of the corresponding topic market data. Assuming that the value is n, then the actual price is (codec reference price) + n* (tick size).
Volume	Vint	Order volume.

When MBP list change events are used to indicate the changes to the MBP list information in instrument snapshots, the integrity of such MBP list information should be ensured. The sequence of MBP list change events as a whole is a transaction, and the Platform only guarantees the information integrity of the MBP list before and after the MBP list change event sequence. When using MBP list change events to reconstruct MBP lists, please strictly follow the order of MBP list change events. Here is an example to show how the Platform generates a MBP list change event and how users should update the MBP list information in snapshots with the MBP list change event sequence.

> As shown in the figure below, after a snapshot of the instrument numbered 20 in a topic with a market depth of 3, the MBP list information on all price levels in the Platform has changed, and the Platform has generated four MBP list change events. MBP list events are coded based on the reference price 23.00 and tick size 0.50.

Instrument 20

	Price Level	Direction	Price	Volume
Ask	4	1	25.50	5
	3	1	25.00	10
	2	1	24.00	20
	1	1	23.50	10
Bid	1	0	22.00	10
	2	0	21.50	15
	3	0	21.00	20
	4	0	20.50	30

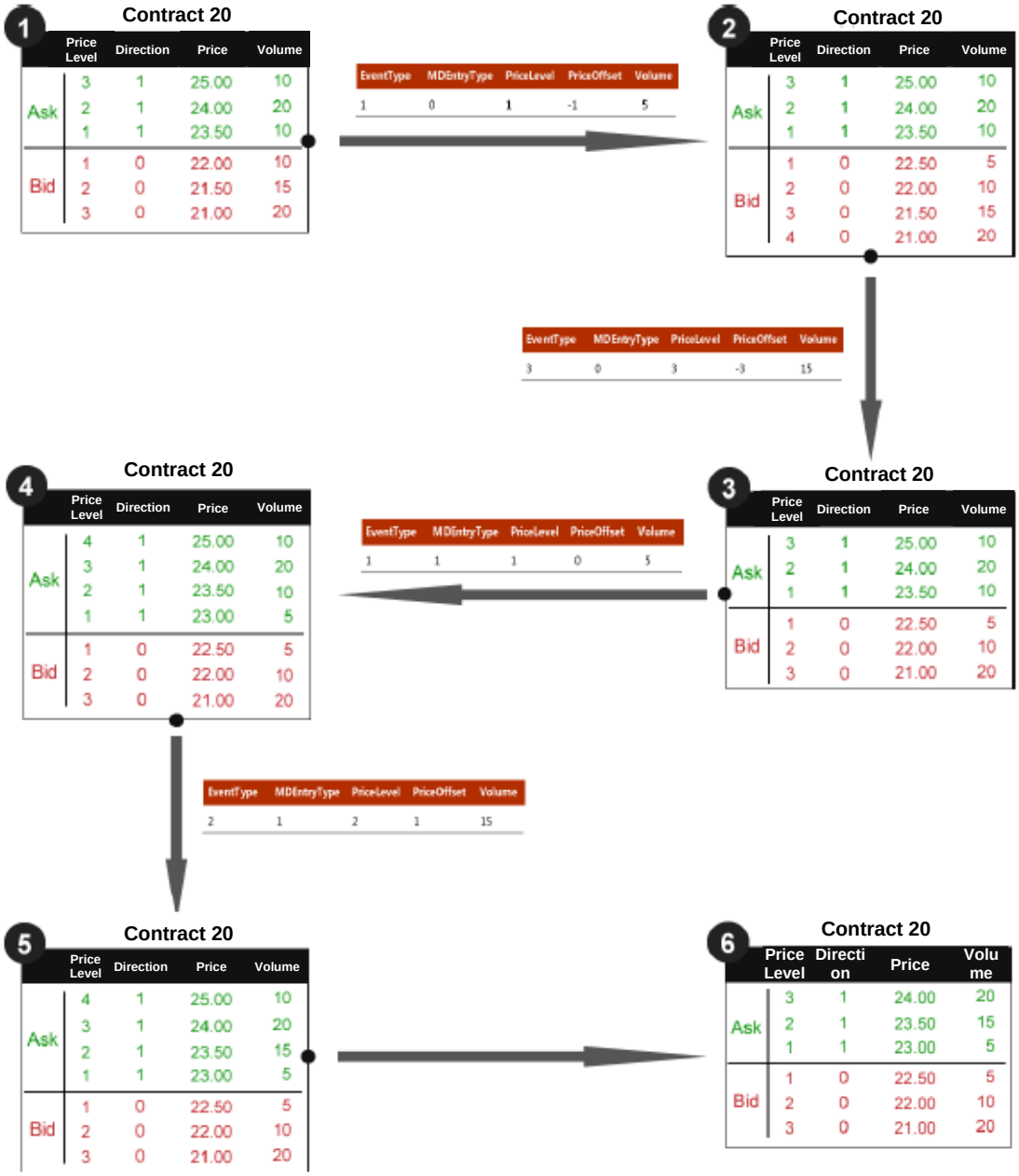
Instrument 20

	Price Level	Direction	Price	Volume
Ask	6	1	25.50	5
	5	1	25.00	10
	4	1	24.50	15
	3	1	24.00	20
	2	1	23.50	15
	1	1	23.00	5
Bid	1	0	22.50	5
	2	0	22.00	10
	3	0	21.00	20
	4	0	20.50	30

EventType	MDEntryType	PriceLevel	PriceOffset	Volume
1	0	1	-1	5
3	0	3	-3	15
1	1	1	0	5
2	1	2	1	15

MBP list Change Event

After receiving the four MBP list change fields sent by the Platform, the client end will sequentially process and update the MBP list with a market depth of 3 owned by corresponding users.



Here are two notes: First, before all the MBP list change fields are processed, do not discard the information beyond the depth of the MBP list, which may be of use later, such as the price level 21.00 in the Bid direction. Second, the information in all MBP list change fields beyond the depth of the MBP list may be inaccurate and cannot be used as a reference, so please do discard it. For example, as shown in the updated MBP list above, the price level 25.00 in the Ask direction has been changed in the Platform.

- **The trade summary field** (FieldID=0x1002) contains the data of a trade summary event. A trade summary event includes the latest trade price, volume and change to open interest in the trades quotation. The event is comprised of data after the change. An instrument incremental contains one trade summary event at most.

Member	Type	Remarks
LastPriceOffset	VInt	The latest price offset. Its coding method and definition are the same as those of the level price in MBP list change events.
VolumeChange	VInt	The change of volume, refers to the difference between the actual trade volume and the trade volume in the previous instrument snapshot. Assume that this value is n, then the trade volume = (original volume) + n.
TurnoverOffset	VInt	The change of turnover. Assume that this value is n, then the actual turnover = (original turnover) + ((volume - original volume) * (codec reference price) + n* (tick size) * (instrument multiplier).

Member	Type	Remarks
OpenInterestChange	VInt	The change of open interest, refers to the difference between the actual open interest and the open interest in the previous instrument snapshot.

- **The highest price field** (FieldID=0x1011) contains the data of a highest price change event. A highest price change event shows the change in the highest price. The event content is the new highest price. An instrument incremental contains one highest price change event at most.

Member	Type	Remarks
HighPriceOffset	VInt	The highest price offset. Its coding method and definition are the same as those of the level price in MBP list change events.

- **The lowest price field** (FieldID=0x1012) contains the data of a lowest price change event. A lowest price change event shows the change in the lowest price. The event content is the new lowest price. An instrument incremental contains one lowest price change event at most.

Member	Type	Remarks
LowPriceOffset	VInt	The lowest price offset. Its coding method and definition are the same as those of the level price in MBP list change events.

- **The open price field** (FieldID=0x1013) contains the data of an open price identification event. An open price identification event provides today's open price in trades quotation, indicating that the open price of the day has been identified. The event content is today's open price. An instrument incremental contains one open price identification event at most. Under normal circumstances, no more than one open price identification event occurs on a trading day.

Member	Type	Remarks
OpenPriceOffset	Vint	The open price offset. Its coding method and definition are the same as those of the level price in MBP list change events.

- **The close price field** (FieldID=0x1014) contains the data of a close price identification event. A close price identification event provides today's close price in trades quotation. The event content is today's close price. An instrument incremental contains one close price identification event at most. Under normal circumstances, no more than one close price identification event occurs on a trading day.

Member	Type	Remarks
ClosePriceOffset	VInt	The close price offset. Its coding method and definition are the same as those of the level price in MBP list change events.

- **The upward limit price field** (FieldID=0x1015) contains the data of an upward limit price change event. An upward limit price change event provides the upward limit price at the moment. The event content is the upward limit price. An instrument incremental contains one upward limit price change event at most. In a static price limit system, the upward limit price is generally unchanged on one trading day; in a dynamic price limit system, there might be more than one changes to the upward limit price on the same trading day.

Member	Type	Remarks
UpperLimitPriceOffset	Vint	The upward limit price offset. Its coding method and definition are the same as those of the level price in MBP list change events.

- **The downward limit price field** (FieldID=0x1016) is used to encapsulate downward limit price change events. A downward limit price change event provides the downward limit price at the moment. The event content is the downward limit price. An instrument incremental contains one downward limit price change event at most. In a static price limit system, the downward limit price is generally unchanged on one trading day; in a dynamic price limit system, there might be more than one changes to the downward limit price on the same trading day.

Member	Type	Remarks
LowerLimitPriceOffset	Vint	The downward limit price offset. Its coding method and definition are the same as those of the level price in MBP list change events.

- **The settlement price field** (FieldID=0x1017) is used to encapsulate settlement price identification events. A settlement price identification event provides today's settlement price. The event content is today's settlement price. An instrument incremental contains one settlement price identification event at most. Under normal circumstances, no more than one settlement price identification event occurs on a trading day.

Member	Type	Remarks
SettlementPriceOffset	Vint	The settlement price offset. Its coding method and definition are the same as those of the level price in MBP list change events.

- **The delta value field** (FieldID=0x1018) encapsulates delta value identification events. A delta value identification event provides today's delta value. The event content is today's delta value. An instrument incremental contains one delta value identification event at most. Under normal circumstances, no more than one delta value identification event occurs on a trading day.

Member	Type	Remarks
CurrDelta	Double	Delta value.

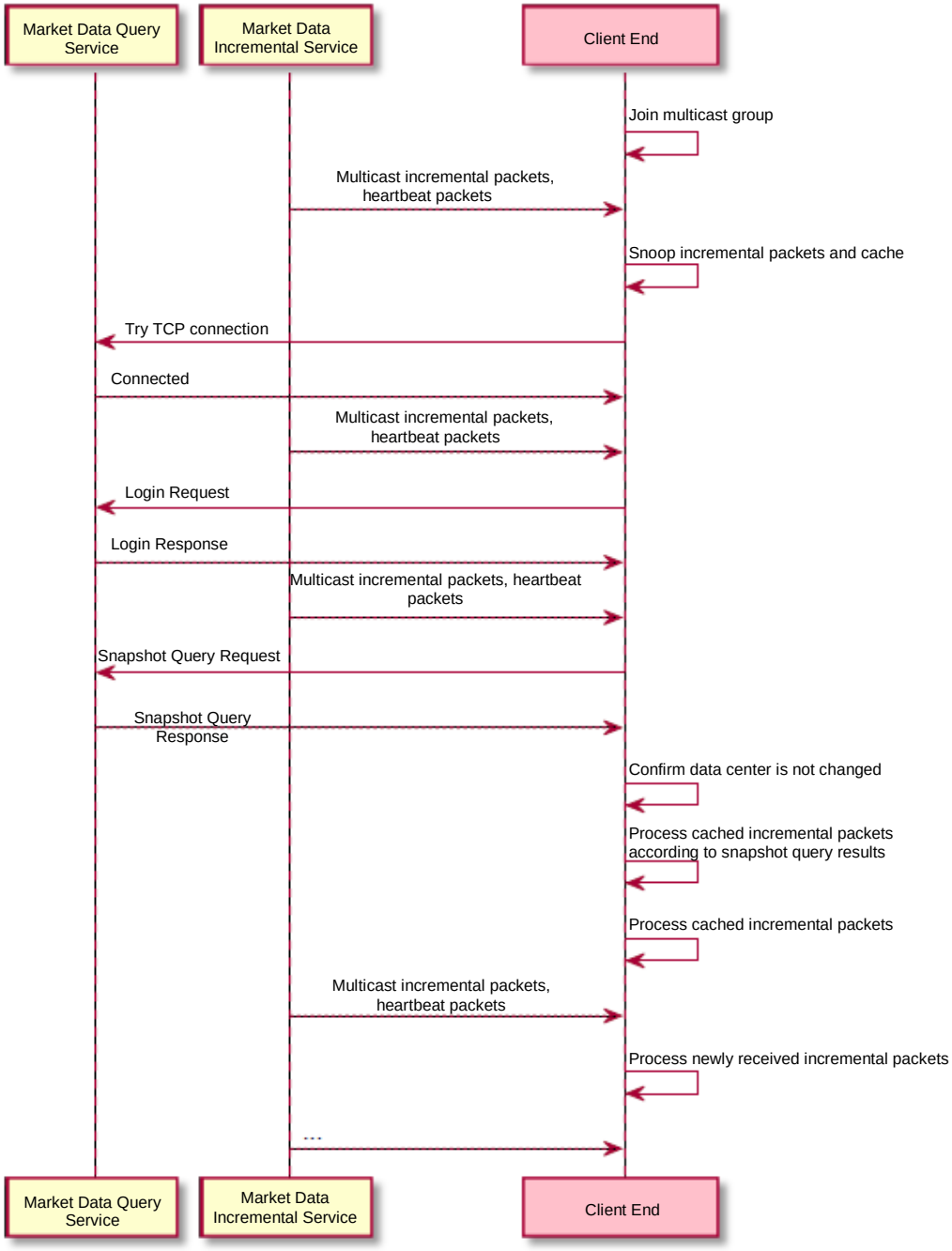
7. Starting and Recovery

This chapter introduces some procedures for the client end to subscribe and process market data with the Platform, including starting subscription, recovering market data and changing data centers.

7.1 Starting

Users shall follow the steps below to connect to the Platform and start the services:

1. **Receive and cache incremental packets:** join the specified multicast group, start receiving MIRP packets, and cache incremental packets; please notice the change of the data center number during the cache: if there is an incremental packet with a larger data center number, discard the original cached packets and re-cache the packets with a larger data center number.
2. **Connect and log in:** establish TCP connection with the market data query service and log in with a login request message.
3. **Query snapshots:** use the topic snapshot query function to query the latest topic snapshot.
4. **Check data center:** Check if the data center number of the topic snapshot to be queried is consistent with the cached incremental packet; if not, return to Step 2.
5. **Process cached incremental packets:** based on the snapshot queried, process cached incremental packets.
    1) Start with the smallest packet number one by one
    2) Discard packets with a number no larger than the latest incremental packet number in the snapshot, until a packet with a number greater than the latest number is found.
    3) Check and confirm that the incremental packet number is the latest number of the incremental packet +1, otherwise, it indicates that the packet is lost; then it is necessary to start market data recovery. (refer to the market recovery section for details)
    4) Process incremental packets in the increasing order of packet numbers.
6. **Keep updating market data:** continue to process newly received incremental packets in the increasing order of packet numbers.



7.2 Market Data Recovery

In some cases, it is necessary for users to recover market data. There are two recovery methods - recovery via snapshot and recovery via incremental supplement.

7.2.1 Recovery via Snapshot

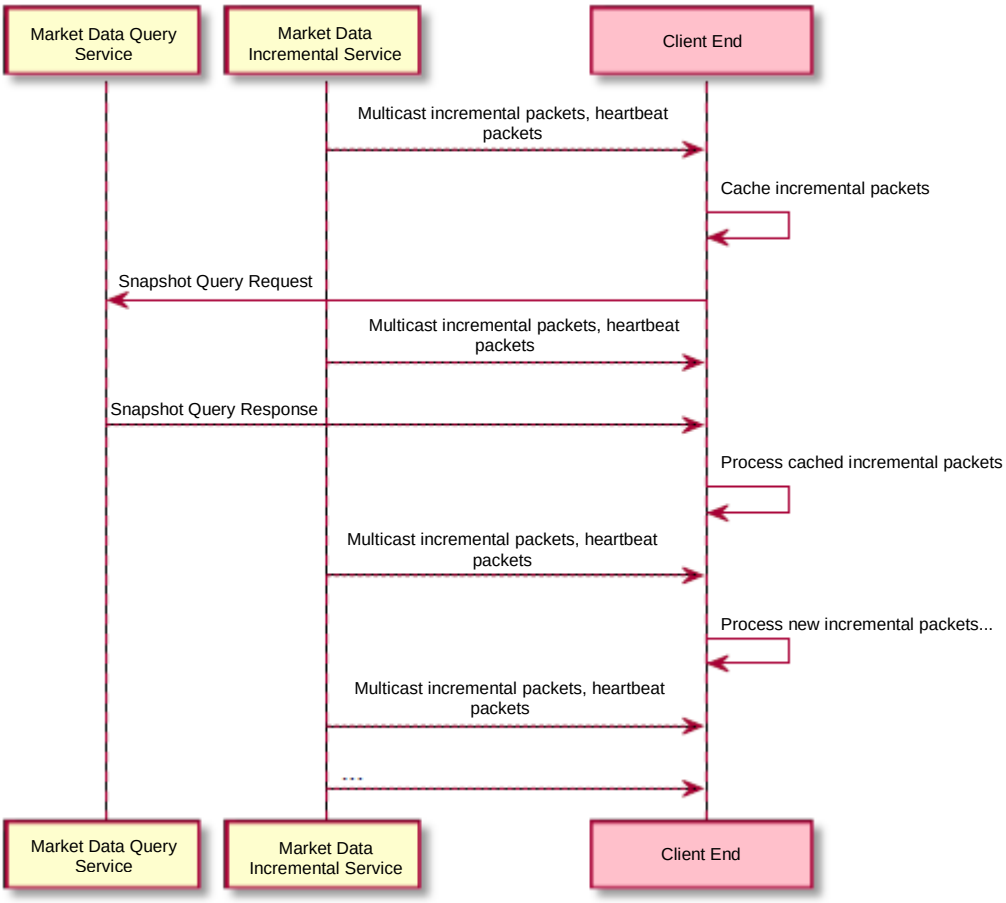
The steps for recovery via snapshot are similar to those for starting: after querying the latest snapshot through the market data query service, please process the incremental packets according to the latest snapshot.

Recovery via snapshot is recommended in the following scenarios:

- Loss of large amounts of incremental packets;
- Heartbeat timeout;
- Change of data centers.

The detailed steps are as follows:

1. Keep receiving MIRP packets and cache all incremental packets; notice the change of the data center number during the cache.
2. Query the latest topic snapshots; if the connection fails, please re-connect and log in first.
3. Check data center.
4. According to the topic snapshot sent from the Platform, sequentially process the cached incremental packets:
    1) Discard packets with a number no larger than the latest incremental packet number in the snapshot.
    2) Check and confirm that the current incremental packet number is the latest number of the incremental packet in the snapshot +1, otherwise, it is necessary to return to Step 2 to restart snapshot query.
    3) Process incremental packets in the increasing order of packet numbers (if the packet body is encrypted, decrypt with the encryption information contained in the topic attributes).
5. Continue to process newly received incremental packets in the increasing order of packet numbers.

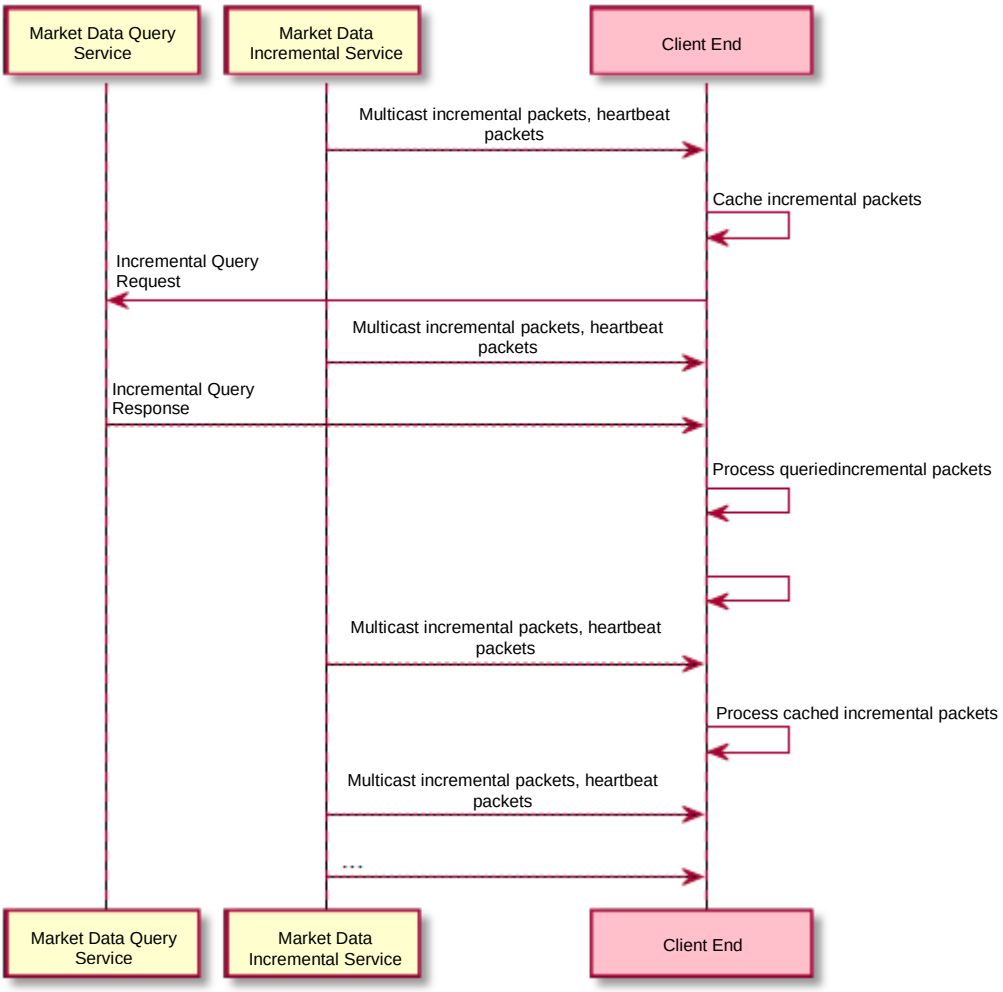


7.2.2 Recovery via Incremental Supplement

Recovery via incremental supplement queries missing incremental packets from the market data query service. This method is recommended for circumstances where only a small number of packets are lost.

The detailed steps are as follows:

1. Keep receiving MIRP packets and cache incremental packets.
2. Identify the number of the missing incremental packet, and send an incremental query request to supplement the corresponding packet.
3. Process in order the incremental packets and cached packets from the Platform contained in its query response message.
4. Continue to process newly received packets.



7.3 Data Center Changing

Where the data center number in the incremental packet received by a user increases, it indicates that the data center has been changed. When receiving such packets, please pay attention to the change of the data center number.
The change between primary and secondary data centers may cause some of the market data that the user has received to be invalid. After discovering that the data center has been changed, the user should try to re-establish the TCP connection with the market data query service and check the latest topic snapshots. Before recovering to the state ready for reception, the user should discard all topic snapshots received with a number greater than the valid snapshot number of the data center change.

Note: Users must join the multicast groups of all centers, otherwise they may not be able to receive new incremental packets after a data center change.

8. Closing Words

This document introduces the protocols and interface specification for the interaction between the Platform and the client end. For any questions, please contact:

> FdTecSys@sfit.shfe.com.cn