Technical Document of Shenzhen Securities Communication Co., Ltd.

SSCC-FDEP User's Manual for Message Transfer System FDEAPI

Document No.:	**FDEP-FDEAPI-PUM001**
Security Classification	**For public use**

深圳证券通信有限公司
SHENZHEN SECURITIES COMMUNICATION CO.,LTD.

April 2020

Document Information

Document Name	FDEAPI User’s Manual
Note	
Project to which it belongs	Financial Data Exchange Platform

Revision History			
Date	**Version**	**Revised by**	**Description**
20190408	1.0	R&D Center	New document
20200117	1.1	R&D Center	Add java interface
20200330	1.2	R&D Center	Add instructions for transferring files
20200403	1.3	R&D Center	Add backup solutions and FAQs

Contents

Figure Index

1 Introduction

This document is the manual for the Financial Data Exchange Application Programming Interface (FDEAPI) of message transmission system, mainly specifies the development methods using FDEAPI, the functions, input, output of each API function, so as to guide developers to use FDEAPI.

The readers of this document include developers, testers, maintenance personnel, etc. who use FDEAPI.

English abbreviation for other terms:

FDEP: Financial Data Exchange Platform

FDSH: FDEP Switching Hub

FDSU: FDEP Switching Unit, component elements of BSSH.

FDAP: FDEP Access Point.

FDMR: Financial Client FDAP, component elements of FDAP.

FPG: Financial Protocol Gateway.

2 Installation and Application Release

2.1 Windows platform

FDEAPI currently supports 32-bit Windows 2003 and 64-bit Windows 2008, Window2016 operating systems. The FDEAPI for the Windows platform consists of the following files:

File name	Description
mr3api.h	Header file for development
mrapi.dll	FDEAPI dynamic link library, used when the application developed by FDEAPI is running.
ini/mrapi.ini	A configuration file in the ini directory.

Use FDEAPI for development. It is recommended to copy all the above files to the same directory of the user's executable file, user can also copy the dll file to the appropriate path and other files to the corresponding path in the development environment.

Please publish mrapi.dll together with mrapi.ini when publishing the FDEAPI application.

2.2 Other platforms

Currently FDEAPI supports the RHEL Server 6.8 operating system of 64-bit and RHEL 7.5 Linux.

FDEAPI can be customized to meet the needs of the user environment to support , for examples AIX 、HP-UNIX and so on to development and use on other platforms.

2.3 Description of configuration

The configuration file ini/mrapi.ini contains some control parameters inside FDEAPI, as well as the generation and output of control logs. The file format is as follows:

```
[CONFIG]
"CompressFileSize"="20000000000" //The file size in bytes. If it is larger than this value, the file will be compressed and then transmitted. If the value is less than or equal to this value, it will be directly transferred without compression. The minimum value allowed to be set by this value is 5120, and the maximum value is not limited.
"DeleteAfterDecompress"="0" //After decompressing the received file, delete the corresponding .bz2 file o rnot, set it to 0-do not delete, other-delete
"ProcessingThread"="5" //Preprocessing thread count, checksum calculation, compression and decompression, default value is 5
"SendBandwith"="1024" //1024 The new version of send queue bandwidth control parameters, in KB/s, this parameter can also be understood as bandwidth, the default value is 1024
"RecvBandwith"="1024" //The new version of send queue bandwidth control parameters, in KB/s, this parameter can also be understood as bandwidth, the default value is 1024
"SendTryTimeInterval"="60" //Interval of sending file retry event in seconds, the default value is 60
"SendTryTimes"="5" // Try times of sending file, the default value is 5
"CompleteTaskTimeout"="5" //Failed, terminated and completed task, the time deleted from the queue, the default value is 5, in minutes
"NotCompressExts"=".bz2.7z.rar.gz.iso.zip" //Do not use the suffix of compressed file, even if the file size exceeds the size set by CompressFileSize, the default value is [].bz2.7z.rar.gz.iso.zip]

"EkeyExtId"="1.2.86.100.4.3.2"//ekey extension field
```

```
"DiffTopicPkgSerial"="10" //Fault tolerance value of the topic message packet sequence number
"SummaryFileMaxSaveDays"="5"//The number of days the task summary file would be saved. It is recommended to keep the same as the maximum number of days of the log MaxSaveDays saved. The default value is -1, not enabled.

 [IpPortMap]//Address Map
"172.100.1.98:3602"="10.10.218.193:3602"
"172.100.1.99:3602"="10.10.218.64:3602"

[AppRecvPath]// Receiving path corresponding to each app
"app1"="C:\\gtest\\app1"
"app2"="C:\\gtest\\app2"
"app3"="C:\\gtest\\app3"

[LOG]
"Type"="1"    //Log type, which ranges from 1 to 2. The default value is 1. 1 indicates cyclic log; 2 indicates log by date.
"LockType"="1"    //The type of log lock, the default value is 1, 1-thread lock, 2-process lock
"Level"="3"       // DEBUG=1, INFO=0, WARN=-1, ERROR=-2，most DEBUG print log, least ERROR print
"Display"="3" //The log output mode is in the range of 0 to 3. The default value is 1. 0 means no log is displayed or recorded; 1 means log is only recorded in file; 2 means log is only displayed on the screen (only valid for console program); 3 means log is recorded in file and displayed on screen simultaneously
"LogDir"="../log" // Directory of log files
"LogName"="mrapi.log" // Prefix of log file name
"MaxFileCount"="20"     // The maximum number of files, the default value is 20
"MaxFileSize"="500000000"     //The size of one of the log files, the default value is 500000000, about 500MB
"MaxSaveDays"="5"    //Maximum days of log saved
```

[Note] There is a hidden parameter "PersistEnable" = "0" in the CONFIG section. The PersistEnable parameter controls the Flag flag when FDEAPI sends a data packet. When there is no configuration item or PersistEnable = 0, Flag disables the MR3_MSGFLAG_PERSIST flag, that is, when sending a packet, the MR3_MSGFLAG_PERSIST flag will be removed by FDEAPI; when PersistEnable = 1, Flag will not disable MR3_MSGFLAG_PERSIST. The default value is "PersistEnable" = "0". The configuration file mrapi.ini hides this configuration item by default..

In the Windows environment, ini/mrapi.ini should be placed in the same directory as mrapi.dll; in the non-Windows platform environment, ini/mrapi.ini should be placed according to the hierarchy after unpacking. If the configuration file mrapi.ini is not in the same-level directory of the mrapi dynamic link library, it is required to set the environment variable "MRAPI_LOGCONF_PATH" as the storage path of the configuration file mrapi.ini. If the configuration file does not exist, or if a parameter is not configured, the corresponding parameter shall be the default value.

During the use of FDEAPI, you can generate a running log and record it in the log file. The [LOG] parameter option of the log in the configuration file can be dynamically modified during the operation of the program using FDEAPI, and it will take effect within 30 seconds after the modification. [CONFIG] parameter options are only valid when FDEAPI is initialized.

3 Application Overview

3.1 Functions

FDEAPI is a set of application programming interfaces (API) that can be called by the user for C language. The user can call the API to develop and conduct data exchange with the message transmission system. FDEAPI can complete the automatic connection of communication, send message packets, receive message packets, encrypt and compress, etc. The application can interact with the access point of the message transmission system using FDEAPI.

3.2 Application environment

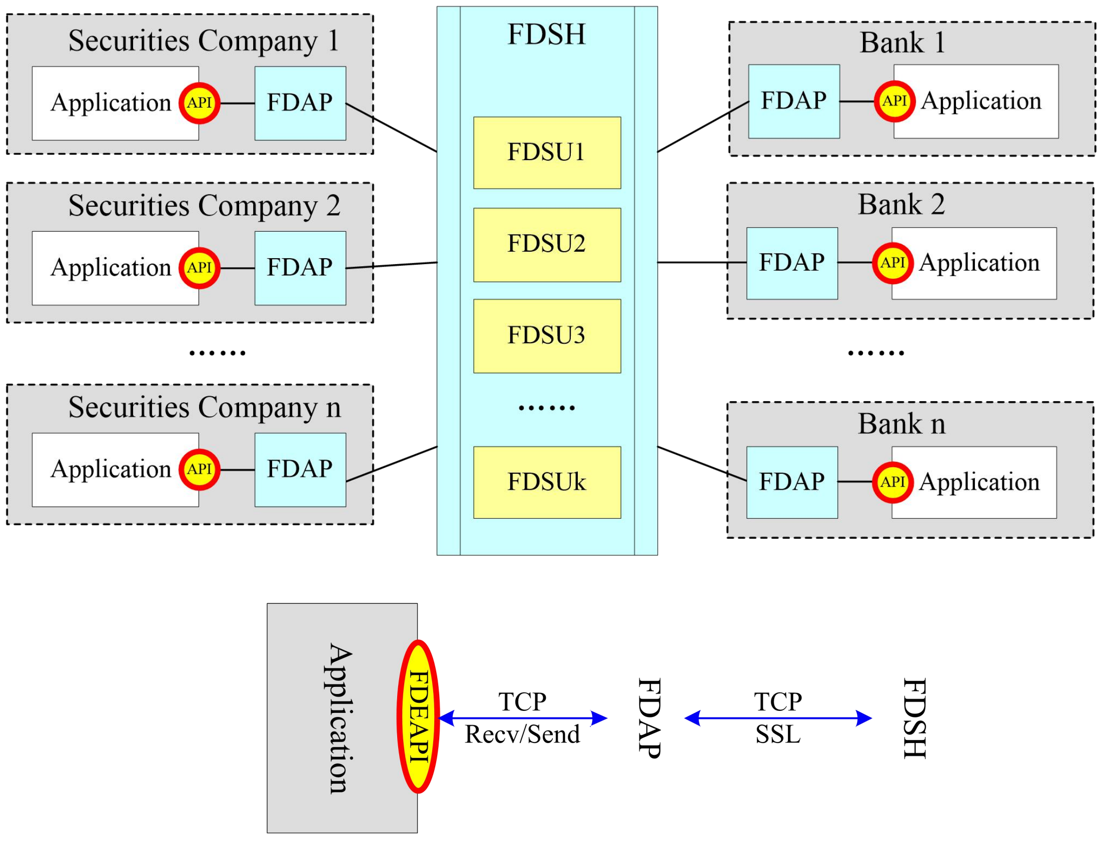


Fig. 1 FDEAPI application environment

In the message transmission system of financial data exchange platform, the access point is used to authenticate the identity of the user for the switching hub, and to establish a secure SSL connection with the switching hub to reliably transmit data. Complex authentication and encryption mechanisms have no longer to be provided for the FDEAPI presented by the access client for users.

3.3 Address ID

The basic function of the financial data exchange platform message transmission system is to transfer messages between users, each message with a source address and a destination address.

Each user of the message transmission system has a unique user ID (UserID), consisting of no more than 63 characters which may be uppercase letters, lowercase letters, numbers, minus signs or underscores. The first character must be a letter. In fact, a user ID corresponds to an access client.

At the user side, an access client can support multiple business applications to access the message transmission system. In order to distinguish different applications, each application has a unique application identifier (AppID) that has the same naming rule as user identifier.

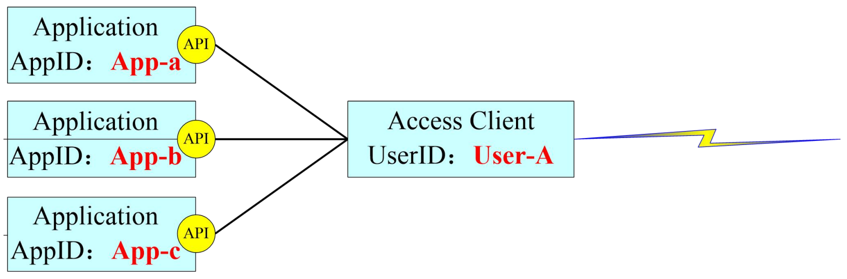


Fig. 2 User ID and application ID

Messages are sent by specific applications and ultimately received and processed by specific applications, both the source address and the destination address of the message should be therefore represented by two elements: the user identifier and the application identifier. A message has four required address elements: source user ID, source application ID, destination user ID, and destination application ID.

3.4 Thread safety

All functions in the FDEAPI are thread-safe. Users can generate multiple threads connected to the client FDAP as needed, and exchange data with the message transmission system to improve switching performance.

3.5 Security of application system

The user may call the FDEAPI in his application software, and there will be no software or code that intentionally attacks the user system in the FDEAPI, and it does not intentionally affect the safe operation of the user system. The security of the user system itself should be checked and maintained by the user since FDEAPI cannot guarantee the security of the user system itself.

4 Programming reference

4.1 Constant definition

4.1.1 Constant of message flag bit

Name	Defined value	Description
MR3_MSGFLAG_PERSIST	0x01	Persistent message flag for reliable transmission, which is currently not enabled.
MR3_MSGFLAG_COMPRESS	0x02	Compression flag, compression transmission is required.
MR3_MSGFLAG_REPLYTOME	0x04	The flag automatically pushed to the sender by response packet.

4.1.2 Length constant

Name	Defined value	Description
MR3_MAXLEN_ADDR	32	The maximum length of user ID and application ID.
MR3_MAXLEN_PKGID	64	The maximum length of message packet ID
MR3_MAXLEN_USERDATA	256	The maximum length of data that the user retains.
MR3_MAXLEN_MSGTYPE	8	The maximum length of the message type identifier.
MR3_MAXLEN_IP	16	IP address length.
MR3_MAXLEN_TOPICNAME	17	The maximum length of topic name, including the ’\0’.

4.1.3 Definition of file status field

```
#define FRAPI_USERSTATUS_WAIT                0  //Wait
#define FRAPI_USERSTATUS_COMPRESS           1// Compression/decompression
#define FRAPI_USERSTATUS_DOING               2   // In transmission
#define FRAPI_USERSTATUS_INHUB               3   // In hub
#define FRAPI_USERSTATUS_DECOMPRESS           4   //Decompress
#define FRAPI_USERSTATUS_COMPLETE            5   //Complete
#define FRAPI_USERSTATUS_FAIL               6   //Fail
#define FRAPI_USERSTATUS_STOPPED             7   //Stop
```

```
#define FRAPI_USERSTATUS_PREPRE            8   //Wait for pretreatment
#define FRAPI_USERSTATUS_CHECKSUM            9   //Check
```

4.1.4 Error value returned by function

For the return value of each function of Mr3Send and Mr3Receive*, returning 0 means success, otherwise it means error, and the error return value has been defined in the mr3api.h file. The function error return value in the mr3api.h file is defined as follows:

```
#define MR3_ERRCODE_OK                        0
#define MR3_ERRCODE_PARAMERR                  -1
#define MR3_ERRCODE_CONNERR                    -2
#define MR3_ERRCODE_TIMEEXPIRED               -3
#define MR3_ERRCODE_TIMEOUT                   -4
#define MR3_ERRCODE_NOMSG                     -5
#define MR3_ERRCODE_BUFTOOSHORT               -6
#define MR3_ERRCODE_BUFTOOBIG                   -7
#define MR3_ERRCODE_SYSERROR                    -8
#define MR3_ERRCODE_COMMU_NOTALLOW     -9
#define MR3_ERRCODE_DEST_NOTONLINE        -10
#define MR3_ERRCODE_DEST_FULL                   -11
#define MR3_ERRCODE_PUBTOPIC_NOTALLOW   -12
```

4.2 Description of data structure

4.2.1 Message attribute STUMsgProperty3 and STUMultiDestMsgProperty3

The STUMsgProperty3 structure is used to represent attributes of a single message, and the STUMultiDestMsgProperty3 structure is used to represent attributes of group sending message.

Structural definition:

```
struct STUMsgProperty3
{
    char                    m_szSourceUserID[MR3_MAXLEN_ADDR];
    char                    m_szSourceAppID[MR3_MAXLEN_ADDR];
    unsigned short          m_usSourceInstID;
    char                    m_szDestUserID[MR3_MAXLEN_ADDR];
```

```
        char                    m_szDestAppID[MR3_MAXLEN_ADDR];
        unsigned short          m_usDestInstID;
        char                    m_szPkgID[MR3_MAXLEN_PKGID];
        char                    m_szCorrPkgID[MR3_MAXLEN_PKGID];
        char                    m_szUserData1[MR3_MAXLEN_USERDATA];
        char                    m_szUserData2[MR3_MAXLEN_USERDATA];
        unsigned char           m_ucFlag;
        unsigned char           m_ucBizType;
        unsigned char           m_ucPriority;
        unsigned char           m_ucSensitiveLevel;
        char                    m_szMsgType[MR3_MAXLEN_MSGTYPE];
        char                    m_szTopicName[MR3_MAXLEN_TOPICNAME];
} ;
```

```
struct STUUserAddr3
{
        char m_szUserID[MR3_MAXLEN_ADDR];
        char m_szAppID[MR3_MAXLEN_ADDR];
} ;

struct STUMultiDestMsgProperty3
{
        char                    m_szSourceUserID[MR3_MAXLEN_ADDR];
        char                    m_szSourceAppID[MR3_MAXLEN_ADDR];
        int                     m_iDestUserCount;
        STUUserAddr3*           m_pArrDestUserAddr;
        char                    m_szPkgID[MR3_MAXLEN_PKGID];
        char                    m_szCorrPkgID[MR3_MAXLEN_PKGID];
        char                    m_szUserData1[MR3_MAXLEN_USERDATA];
        char                    m_szUserData2[MR3_MAXLEN_USERDATA];

        unsigned char           m_ucFlag;
        unsigned char           m_ucBizType;
        unsigned char           m_ucPriority;
        unsigned char           m_ucSensitiveLevel;
        char                    m_szMsgType[MR3_MAXLEN_MSGTYPE];
} ;
```

Field description:

Field	Description

Field	Description
m_szSourceUserID	Source user ID, character string ending with "\0".
m_szSourceAppID	Source application ID, character string ending with "\0".
m_usSourceInstID	Source instance ID, greater than 0, less than 65535.
m_szDestUserID	Destination user ID, character string ending with "\0".
m_szDestAppID	Destination application ID, character string ending with "\0".
m_usDestInstID	Destination instance ID, greater than 0, less than 65535.
m_szPkgID	The packet identifier of the message bundle, a character string ending with "\0", or generated by the user calling the MrCreatePkgID function, or empty (ie '\0').
m_szCorrPkgID	Related package identifier, a character string ending with "\0" for user.
m_szUserData1	User data 1, a character string ending with "\0" for the user.
m_szUserData2	User data 2, a character string ending with "\0" for the user.
m_ucFlag	The message flag consists of 8 binary digits. The meanings of bits are as follows: Bit 0 -- 1 means persistent messages, which need to be transmitted reliably, currently not supported; Bit 1 -- 1 means the message should be compressed before transmission; Bit 2 -- 1 means that the response packet is automatically pushed to the sender.
m_ucBizType	Service type flag: 0 to 255.
m_ucPriority	Priority level flag: 3 to 5, 5 is the lowest, and 3 is the highest.
m_ucSensitiveLevel	Sensitive level flag: 0 to 255, 0 is the lowest and 255 the highest.
m_szMsgType	The message type identifier, the first character 'M' means message and 'F' means file.
m_iDestUserCount	Number of designation address
m_pArrDestUserAddr	An array pointer containing m_iDestUserCount destination addresses.

Special notes:

The three fields m_szCorrPkgID, m_szUserData1, and m_szUserData2 are for the user to use. The message transmission system itself does not use these fields, nor does it actively change them.

The compression and decompression of the message packet is automatically completed by the message transmission system, transparent to the user. Setting bit 0 of m_ucFlag to 1 is to notify the message transmission system to reliably transmit the message; Setting bit 1 of m_ucFlag to 1 is to notify the financial data exchange platform to transmit the message after compression, and the message packet available to the user via the FDEAPI is always uncompressed; setting bit 2 of m_ucFlag to 1 is to inform the message transmission system to automatically push the response packet to the sender.

Constraints**:**

If the user calls the Mr3Send function to send business response packet, for the m_szCorrPkgID field in the pMsgPropery parameter, the unified value to be filled shall be the same value as the m_szPkgID field in the corresponding request packet, so that the requester can receive the response packet according to the field and find the corresponding request packet. This field shall be empty (ie '\0') for business request packets or other types of packets.

4.2.2 Linkage information STUConnInfo3

This structure is used to define the information needed to establish a connection with access client.

Structural definition:

```
struct STUConnInfo3
{
        char                    m_szMRIP[MR3_MAXLEN_IP];
        unsigned short          m_usMRPort;
};
```

Field description:

Field	Description
m_szMRIP	The IP address of theAccess client FDAP, which is a character string ending in "\0", in the format of "xxx.xxx.xxx.xxx".
m_usMRPort	Connect to the connection port of the Access client FDAP.

Special notes:

This structure is used as an input parameter when calling the Mr3Init function. The user shall pass in the array pointer and the number of array elements of the structure. Mr3Init first tries to connect to the first client FDAP specified in the structure array and if unsuccessful, tries to connect to the next client FDAP of the array. The access client consists of at least two client FDAPs, and more client FDAPs are also acceptable. When using FDEAPI, as long as you connect one of the specified client FDAPs, the load between all the client FDAP of the access client will be automatically balanced, that is, the client FDAP to which the application ultimately connect may not be any of the specified ones in the structure.

4.2.3 App status information STUAppStatus

STUAppStatus Structure is used to represent the status information of a single instance of app.

Structure definition:

```
STUAppStatus
{
        char    m_szAppID[MR3_MAXLEN_ADDR];     /*Source application ID, must
be a string ending in '\0' */
        char m_szLinkID[MR3_MAXLEN_ADDR];       /*Application connection ID,
a string ending with '\0' */
```

```
        unsigned short m_usInstanceID;      /* Application instance ID,    a string ending
with '\0' */
        char    m_szIP[MR3_MAXLEN_IP];    /*Application example server IP address,
a string ending with '\0' */
        unsigned short      m_usPort;       /*Application instance service port number */
        char       m_szMrName[MR3_MAXLEN_ADDR];          /*Application   instance
login MR name*/
        char          m_szLoginTime[MR3_MAXLEN_USERDATA];     /*The time of the
application instance logs on to MR */
        long  long  m_i64QueueLength;         /*Receive  queue  length  of  application
identification */
        long long     m_i64TopicQueueLength;    /*Receive topic message queue length
of application identification*/
        long long      m_i64FileQueueLength;    /*Receive file message queue length of
application identification */
        long long     m_i64SendPkgCountTotal;    /*Total number of packets sent by the
application instance(this login) */
        long long    m_i64RecvPkgCountTotal;    /*Total number of packets received by
the application instance(this login) */
} ;
```

4.3 File transmission

4.3.1 Transmission architecture

The file transfer function is implemented on the basis of the message exchange function of the message transmission system. The sending program only needs to call the API function of sending the file. The API internally transfers the file content to the receiving end through the message to realize the file sending function. The receiving program only needs to poll the API function of the receiving file to receive the complete file content, as shown in the figure below.

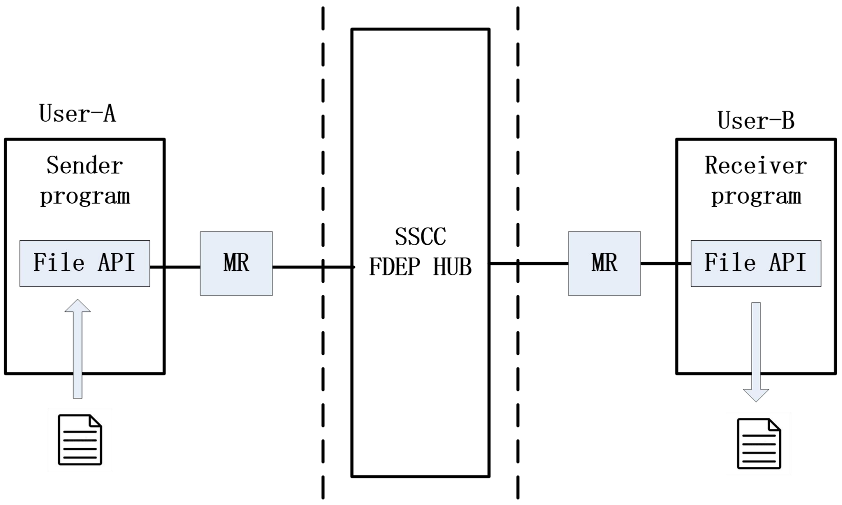


Fig. 3 Schematic diagram of FDEAPI sending / receiving files

4.3.2 Implementation mechanism

The user first needs to call Mr3Init for initialization, ensure that the client BSMR is connected through Mr3IsLinkOK, and then start the file transfer.

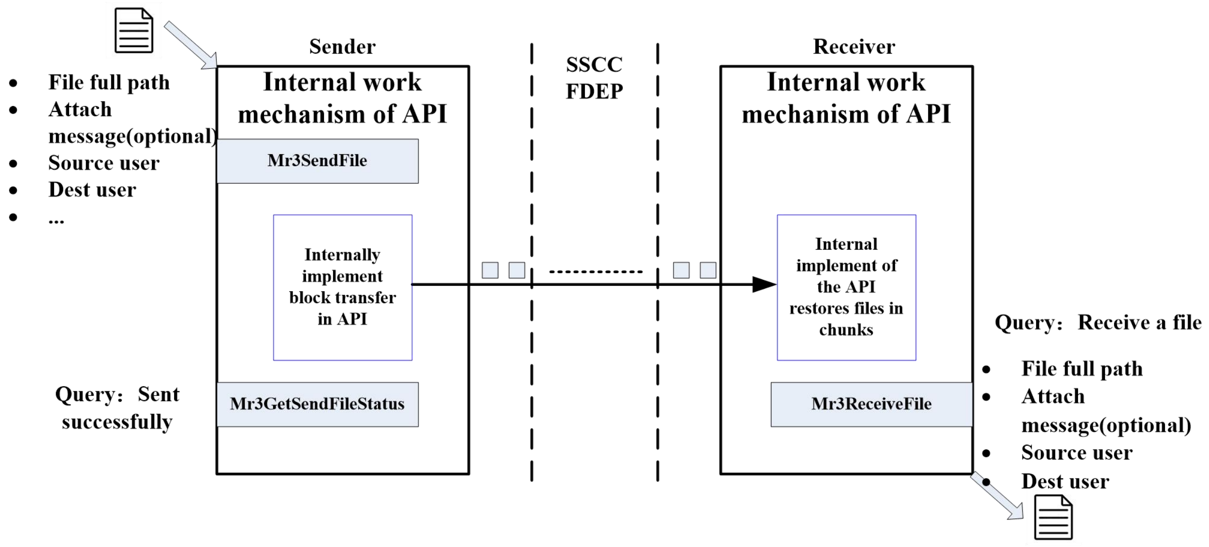


Fig. 4 FDEAPI file transfer mechanism

Sender internal processing：

- The user calls the Mr3SendFile interface to pass in the full path of the send file, sender user ID, sender APPID, sender instance ID, receiver user ID, receiver APPID, receiver instance ID, and other file attribute (optional) information；

- After checking and compressing the files in the API program, the files are sent to the receiver in blocks;
- If the user needs to get the status of the file sent, he can poll MR3GetSendFileStatus to get the status of the file sent.

Receiver internal processing:

- The receiving end belongs to a passive processing mode. When the receiving end API receives a file block internally, it will continue to receive the file block at a break point, and assemble the file block into a complete file, decompress it, verify the processing, and then generate Receiving the completion notification message to inform the receiving API function;
- The receiving end user polls and calls MR3ReceiveFile to check whether a file has been received. The interface call returns success, indicating that the file has been received completely and stored in the receiving path, and the interface returns other related attribute information.

4.3.3 Calling sequence

Timing diagram for both sides:

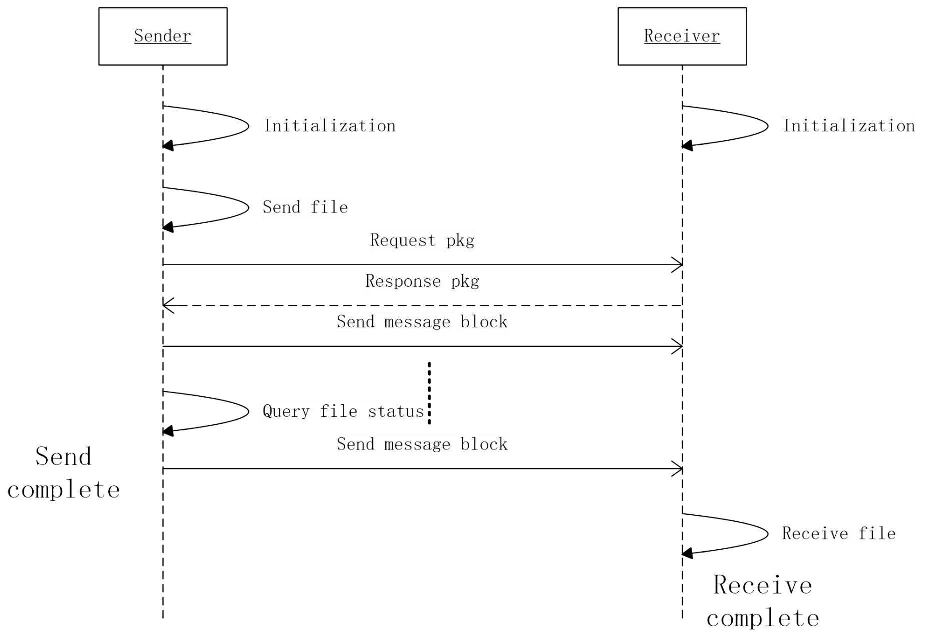
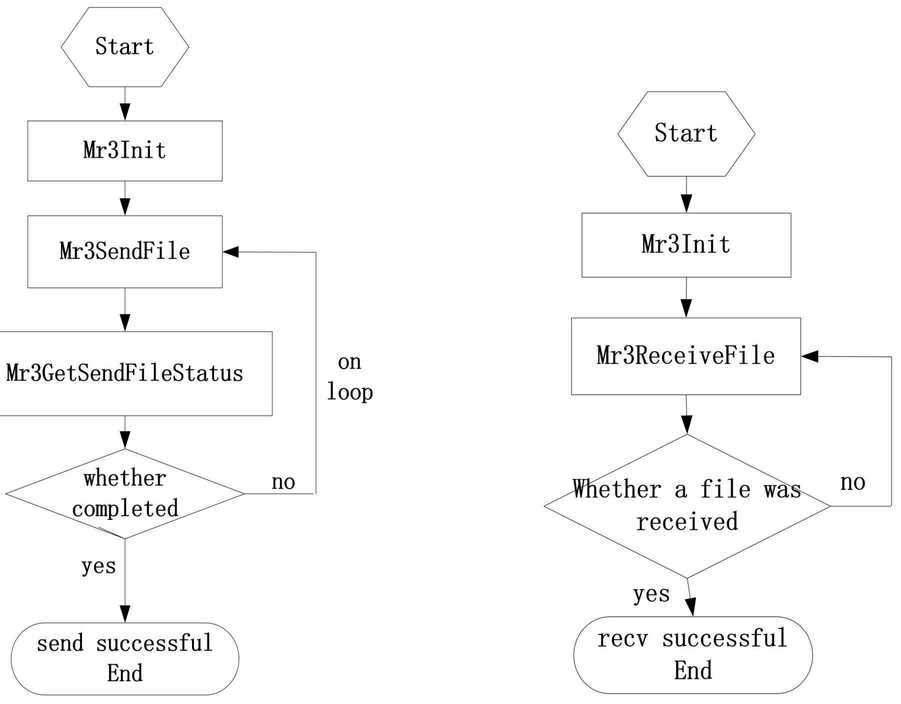


Fig. 5 Timing chart of file transfer between two parties

Send and receive file call flowchart is shown below：



4.3.4 Application Notes

1、The file transfer interface is relatively simple. It only contains 4 functions (send 3 functions and receive 1 function). The complex file data transmission process has been encapsulated in the interface. It is transparent to API developers, and users do not need to pay attention to file block and break. Click to resume, compression / decompression, integrity check and other complex processes.

2、File transfer is based on the online transmission mode of both parties, and requires the sender and receiver to be online at the same time.

3、File transfer is implemented based on the message transmission system. It shares the same line and common bandwidth as ordinary messages. In practical applications, the bandwidth allocation of common messages and file data needs to be considered. The bandwidth control of file transmission is configured in mrapi.ini. The appropriate bandwidth can be allocated to file transmission according to the physical bandwidth situation.

4、The file transfer interface is a new feature after FDEPv5. The sender and receiver must be versions after v5 to support the file transfer function.

4.4 C function interface

4.4.1 Function list

FDEAPI is a concise and easy-to-use programming interface that provides the following 20 functions:

No.	Function name	Functional features
1	Mr3Init	Initialize, get the relevant resources, and try to establish a connection with the access client FDAP.
2	Mr3IsLinkOK	Check and judge whether the current connection with the access client FDAP is normal.
3	Mr3CreatePkgID	Generate a message packet ID.
4	Mr3Send	Send message individually to the message hub via FDAP, request forwarding.
5	Mr3MultiDestSend	The message is sent in a group to the message hub through the FDAP, and split into multiple independent messages in the hub for processing and forwarding.
6	Mr3Receive1	The message forwarded by the message hub is received in the mode 1 condition.
7	Mr3Receive1_FreeBuf	Release the memory allocated in the Mr3Receive1 function call.
8	Mr3Receive2	The message forwarded by the message hub is received in the mode 2 condition.
9	Mr3Receive3	The message forwarded by the message hub is received in the mode 3 condition.
10	Mr3SendTopicMsg	Send message individually to the message hub via FDAP, request forwarding.
11	Mr3ReceiveTopicMsg	Receive the topic message forwarded by the hub.
12	Mr3SendFile	Send message individually to the message hub via FDAP, request forwarding.
13	Mr3ReceiveFile	Receive the file message forwarded by the hub.
14	Mr3Destroy	Disconnect from the FDAP and release relevant resources.
15	Mr3GetVersion	Get the version number of the API.
16	Mr3RegRecvCondition	Registration package push-down conditions, push all conditions at once.
17	Mr3GetPeerUserStat	Get the status of the communication peer user.
18	Mr3CancelSendFile	Cancel sending files.

No.	Function name	Functional features
19	Mr3GetSendFileStatus	Get the status and progress of the file sent.
20	Mr3GetAppStatus	Get app status information

Note: The MRAPI dynamic link library also contains the functions beginning with MR/MR2, which are reserved for compatibility with the old version of the interface; the function beginning with MR3 is a new interface. Old interface function cannot not be mixed with new ones.

4.4.2 Mr3Init

The initialization function when connecting to FDAP can initialize the FDEAPI, allocate the relevant resources, and attempt to establish a communication connection with the access client.

When FDEAPI calls this initialization function, it will try to connect to the client using TCP. In general, the connection will succeed before the function is returned. If the access client is temporarily unavailable or there is any network problem, the access client may not be connected when the calling function returns. Because the FDEAPI has the function of reconnection in case of disconnection, the API will automatically retry connection with the counterparty.

Function prototype:

```
void* _stdcall Mr3Init(const char* psAppID, unsigned short usInstID, const char* psAppPasswd, OnReceiveCallBack3 onReceive, const STUConnInfo3* pArrConnInfo3, int iArrConnInfoCount, void* pvUserData);
```

Parameter description:

Parameter	Description
psAppID [in]	Application ID of the application
usInstID[in]	Instance ID
psAppPasswd[in]	The password set by the application at the access client, the password must match the preset one.
OnReceive[in]	The callback function when receiving message packet. The callback function shall not be used simultaneously with the Mr3Receive1/Mr3Receive1_FreeBuf, Mr3Receive2 or Mr3Receive3/Mr3Receive1_FreeBuf.
pArrConnInfo3[in]	Connection information array at access client.
iArrConnInfoCount [in]	Number of connection information array element at access client

pvUserData [in]	User data to be used by callback function

Description of return value:

Return value	Description
NULL	Initialization failed.
Non-NULL	Successful initialization, returning a connection handle that will be used as a parameter called by other functions.

4.4.3 Mr3IsLinkOK

Function for connection status judgment. This function determines whether the current connection with the access client is normal. Since the message exchange performed by the entire financial data exchange platform is asynchronous and the FDEAPI has the function of automatic reconnection, abnormal connection would not indicate that the system does not work normally, and after the connection is automatically restored, tasks will continue.

Note: Mr3Init returning available handles shall not mean that the network has been successfully connected, possibly the network connection is in progress and, to avoid misjudgment, it is required to wait for a few seconds before making the Mr3IsLinkOK judgment after the Mr3Init function is called.

Function prototype:

```
int _stdcall    Mr3IsLinkOK(void* pHandle);
```

Parameter description:

Parameter	Description
pHandle [in]	Connect handle, the value returned when calling Mr3Init.

Description of return value:

Return value	Description
0	Abnormal connection
1	Normal connection

4.4.4 Mr3CreatePkgID

Message packet ID generates function. This function can generate a globally unique UUID throughout FDEP that can be used as a message packet identifier.

Function prototype:

```
Int _stdcall Mr3CreatePkgID(void* pHandle, char szPkgID[MR3_MAXLEN_PKGID]);
```

Parameter description:

Parameter	Description
pHandle [in]	Connect the handle, the return value when calling Mr3Init.
szPkgID [out]	Messag package identifier generated

Description of return value:

Return value	Description
0	Successful
Others	Failed

4.4.5 Mr3Send

Single-sending message packet function can perform cthe function of sending a message to a single destination user.

Appointment:

If the user calls the Mr3Send function to send business response packet, for the m_szCorrPkgID field in the pMsgPropery parameter, the value same as the m_szPkgID field in the corresponding request packet shall be filled in, so that the requester can receive the response packet according to the field and find the corresponding request packet. For business request packets or other types of packets, this field shall be empty (ie '\0').

Function prototype:

```
int _stdcall   Mr3Send(void* pHandle, const char* psPkg, int iPkgLen, STUMsgProperty3 *pMsgProperty, int iMillSecTimeo);
```

Parameter description:

Parameter	Description
pHandle [in]	Connection handle, the value returned when calling Mr3Init.
psPkg [in]	The message packet buffer to be sent.
iPkgLen [in]	The length of message packet in the buffer.
pMsgPropery [in/out]	Message packet attribute. (1) The input of m_szSourceUserID is ignored, and the output is the user ID of the user set in the access client. (2) The input of m_szSourceAppID is ignored, and the output is the application ID of the application set in Mr2Init. (3) m_szDestUserID and m_szDestAppID input are the destination address of the message packet - the destination user ID and the destination application ID, and the output is unchanged.

Parameter	Description
	(4) The m_szPkgID input is a unique message packet ID. This value is either a unique identifier generated by the user calling the MrCreatePkgID function, and the output is unchanged; or the input is empty (ie, '\0'), then the output is the unique message packet ID automatically assigned by the system. For each packet sent, the m_szPkgID should be unique across the entire financial data exchange platform, and different packages should use different m_szPkgIDs. Although the message ID generated by the Mr2CreatePkgID function is unique, different users may take different calling methods because of different programming methods, some may not call the message by standard methods, or use it for special purposes by adding some extra information, even resulting in programming errors, etc., thus m_szPkgID or m_szCorrPkgID may be repeated. Users therefore cannot simply use m_szPkgID or m_szCorrPkgID as the globally unique ID. FDEP will only correctly transmit the packet to the destination, and the other party should, after receiving the message packet, perform different business processes depending on different situations, e.g. checking user serial number, fund account number, user name and other information, and determining the package related information by checking the business information. (5) The m_szCorrPkgID, m_szUserData1, and m_szUserData2 inputs are user-defined data, and the output is unchanged. (6) The m_ucFlag input is the required packet processing bit flag, and the output is unchanged. (7) m_ucBizType input refers to the business type of the data. (8) m_ucPriority input refers to the priority level of the message transmission. (9) m_ucSensitiveLevel input refers to the sensitivity level of the message content. (10) The m_szMsgType input refers to the type of the message.
iMillSecTimeo [in]	The maximum timeout for sending in milliseconds.

Description of return value:

Return value	Description
0	Successful
Others	Failed

4.4.6 Mr3MultiDestSend

Group-sending message packet function can perform the function of sending message to multiple destination users. Group-sending message does not support sending reliable message.

Function prototype:

```
int _stdcall   Mr3MultiDestSend(void* pHandle, const char* psPkg, int iPkgLen,
STUMultiDestMsgProperty3* pMsgPropery);
```

Parameter description:

Parameter	Description
pHandle [in]	Connection handle, the value returned when calling Mr3Init.
psPkg [in]	The message packet buffer to be sent.
iPkgLen [in]	The length of message packet in the buffer.
pMsgPropery [in/out]	Message packet attribute. (1) m_iDestUserCount input refers to the number of destination addresses to be sent. (2) m_pArrDestUserAddr input refers to the array pointer, the array contains multiple destination address structures. The other fields are the same as the fields of Mr3Send's pMsgPropery.

Description of return value:

Return value	Description
0	Successful
Others	Failed

4.4.7 Mr3Receive1

Message packet receiver function 1: The first eligible message is received from the receiving queue corresponding to the application according to the given condition, and removed from the queue. The memory occupied by the message packet received by this function is allocated internally by the FDEAPI. After use, the Mr3Receive1_FreeBuf function should be called to release the allocated memory.

Note: When receiving a message packet, either the callback function OnReceive in the function Mr3Init, or Mr3Receive1/Mr3Receive1_FreeBuf, Mr3Receive2 or

Mr3Receive3/Mr3Receive1_FreeBuf can be used for reception purpose, but not both.

Function prototype:

```
int _stdcall   Mr3Receive1(void* pHandle, char** ppsPkg, int* piOutPkgLen,
STUMsgProperty3 *pMsgProperty, int iMillSecTimeo);
```

Parameter description:

Parameter	Description
pHandle [in]	Connection handle, the value returned when calling Mr3Init.
ppsPkg [out]	A dual pointer, the memory pointed to by the return package. This memory is allocated inside the function. After using the memory, the user has to call the Mr3Receive1_FreeBuf function to release the memory.
piOutPkgLen [out]	The actual length of the message packet received.
pMsgPropery [in/out]	The input will be considered as the receiving condition, and six fields in the structure are used as the judgment conditions: m_szSourceUserID, m_szSourceAppID, m_szPkgID, m_szCorrPkgID, m_szUserData1, and m_szUserData2. Other fields are ignored. If one of the input values in the six fields is empty, the field is also excluded from the judgment condition, that is, the actual judgment condition is a non-empty field among the six fields. The criteria for compliance are that all corresponding fields in the message attribute are the same as these non-empty condition field values. If the 6 fields are all empty, the first message packet in the queue is received. Note that there are special treatments for the input condition of m_szCorrPkgID: when the value of the m_szCorrPkgID string is empty, it can match the message with the m_szCorrPkgID field as an arbitrary value; when the value of the m_szCorrPkgID string is "<EMPTY>", then it only matches the message with m_szCorrPkgID field being empty; when the value of the m_szCorrPkgID string is "<NOEMPTY>", it only matches the message with m_szCorrPkgID field being non-empty. The output is the attribute of the packet received.
iMillSecTimeo [in]	The maximum timeout for receiving in milliseconds. This time is the maximum time interval between sending packet request and receiving or not receiving a response by the function for internal control purpose, rather than the actual time interval at which the function actually executes. For example, if there is no eligible packet at this time, even if the value is 1 minute, the function will return the packet immediately as soon as the response is received, unnecessary to wait until 1 minute before returning. This value is related to the busy program of the system. If the system is busy, the value can be appropriately larger. If the system is idle, the value may be appropriately smaller. In general, the value is recommended to be 2,000, namely 2 seconds.

Description of return value:

Return value	Description
0	Successful
Others	Failed

4.4.8 Mr3Receive1_FreeBuf

Message packet memory free function: This function frees message packet memory that is automatically allocated internally by FDEAPI when the Mr3Receive1 or Mr3Receive3 function is called.

Function prototype:

```
void _stdcall    Mr3Receive1_FreeBuf(char* psPkg);
```

Parameter description:

Parameter	Description
psPkg [in]	The pointer returned by the second parameter of the Mr3Receive1 or Mr3Receive3 function.

Description of return value:

Return value	Description
No	

4.4.9 Mr3Receive2

Message packet receiver function 2: This function has functions similar to the Mr3Receive1, but user has to allocate and manage memory for the message receiving buffer. In addition, the user shall allocate a large enough memory in advance and then call this function to receive message.

Note: When receiving a message packet, either the callback function OnReceive in the function Mr3Init, or Mr3Receive1/Mr3Receive1_FreeBuf, Mr3Receive2 or Mr3Receive3/Mr3Receive1_FreeBuf can be used for reception purpose, but not both.

Function prototype:

```
int _stdcall   Mr3Receive2(void* pHandle, char* psPkg, int* piOutPkgLen, int iBufLenIn, STUMsgProperty3 *pMsgProperty, int iMillSecTimeo);
```

Parameter description:

Parameter	Description
pHandle [in]	The connection handle, the value returned when calling Mr2Init.
psPkg [out]	Message packet receive buffer.
piOutPkgLen [out]	The actual length of the message packet received.
iBufLenIn [in]	Size of message packet buffer.
pMsgPropery [in/out]	The input will be considered as the receiving condition, and six fields in the structure are used as the judgment conditions: m_szSourceUserID, m_szSourceAppID, m_szPkgID, m_szCorrPkgID, m_szUserData1, and m_szUserData2. Other fields are ignored. If one of the input values in the six fields is empty, the field is also excluded from the judgment condition, that is, the actual judgment condition is a non-empty field among the six fields. The criteria for compliance are that all corresponding fields in the message attribute are the same as these non-empty condition field values. If the 6 fields are all empty, the first message packet in the queue is received. Note that there are special treatments for the input condition of m_szCorrPkgID: when the value of the m_szCorrPkgID string is empty, it can match the message with the m_szCorrPkgID field as an arbitrary value; when the value of the m_szCorrPkgID string is "<EMPTY>", then it only matches the message with m_szCorrPkgID field being empty; when the value of the m_szCorrPkgID string is "<NOEMPTY>", it only matches the message with m_szCorrPkgID field being non-empty. The output is the attribute of the packet received.
iMillSecTimeo [in]	The maximum timeout for receiving in milliseconds. This time is the maximum time interval between sending packet request and receiving or not receiving a response by the function for internal control purpose, rather than the actual time interval at which the function actually executes. For example, if there is no eligible packet at this time, even if the value is 1 minute, the function will return the packet immediately as soon as the response is received, unnecessary to wait until 1 minute before returning. This value is related to the busy program of the system. If the system is busy, the value can be appropriately larger. If the system is idle, the value may be appropriately smaller. In general, the value is recommended to be 2,000, namely 2 seconds.

Description of return value:

Return value	Description
0	Successful
Others	Failed

4.4.10 Mr3Receive3

Message packet receiver function 3: This function has functions similar to the Mr3Receive1, however it can receive system error messages. When the target address does not exist, the queue is full, the timeout expires, the system shows errors, etc., for example, the value of piErrSXCode parameter of the function will be non-zero, and the value indicates the reason code of the error, and the error reason string in the ppsPkg parameter; when the value of piErrSXCode parameter is 0, it indicates that the normal message packet is received. The memory occupied by the message packet received by this function is allocated internally by the FDEAPI. After use, the Mr3Receive1_FreeBuf function should be called to release the allocated memory.

Note: When receiving a message packet, either the callback function OnReceive in the function Mr3Init, or Mr3Receive1/Mr3Receive1_FreeBuf, Mr3Receive2 or Mr3Receive3/Mr3Receive1_FreeBuf can be used for reception purpose, but not both.

Function prototype:

```
int _stdcall Mr3Receive3(void* pHandle, char** ppsPkg, int* piOutPkgLen, int* piErrSXCode, STUMsgProperty3 *pMsgProperty, int iMillSecTimeo);
```

Parameter description:

Parameter	Description
pHandle [in]	Connection handle, the value returned when calling Mr3Init.
ppsPkg [out]	A dual pointer, the memory pointed to by the return package. This memory is allocated inside the function. After using the memory, the user has to call the Mr3Receive1_FreeBuf function to release the memory.
piOutPkgLen [out]	The actual length of the message packet received.
piErrSXCode[out]	Reason code of switching error: when the value is 0, it means that the normal exchange message packet is received; when the value is 1 to 5, it means that the target does not exist, there is a target error, the queue is full, the timeout expires or the system

Parameter	Description
	shows error, then *ppsPkg is the error string, and pMsgPropery is the parameter when the original packet is sent.
pMsgPropery [in/out]	The input will be considered as the receiving condition, and six fields in the structure are used as the judgment conditions: m_szSourceUserID, m_szSourceAppID, m_szPkgID, m_szCorrPkgID, m_szUserData1, and m_szUserData2. Other fields are ignored. If one of the input values in the six fields is empty, the field is also excluded from the judgment condition, that is, the actual judgment condition is a non-empty field among the six fields. The criteria for compliance are that all corresponding fields in the message attribute are the same as these non-empty condition field values. If the 6 fields are all empty, the first message packet in the queue is received. Note that there are special treatments for the input condition of m_szCorrPkgID: when the value of the m_szCorrPkgID string is empty, it can match the message with the m_szCorrPkgID field as an arbitrary value; when the value of the m_szCorrPkgID string is "<EMPTY>", then it only matches the message with m_szCorrPkgID field being empty; when the value of the m_szCorrPkgID string is "<NOEMPTY>", it only matches the message with m_szCorrPkgID field being non-empty. The output is the attribute of the packet received.
iMillSecTimeo [in]	The maximum timeout for receiving in milliseconds. This time is the maximum time interval between sending packet request and receiving or not receiving a response by the function for internal control purpose, rather than the actual time interval at which the function actually executes. For example, if there is no eligible packet at this time, even if the value is 1 minute, the function will return the packet immediately as soon as the response is received, unnecessary to wait until 1 minute before returning. This value is related to the busy program of the system. If the system is busy, the value can be appropriately larger. If the system is idle, the value may be appropriately smaller. In general, the value is recommended to be 2,000, namely 2 seconds.

Description of return value:

Return value	Description
0	Successful
Others	Failed

4.4.11 Mr3SendTopicMsg

Send individual topic message packet function. This function sends message to a single destination user.

Function prototype:

```
int _stdcall   Mr3SendTopicMsg(void* pHandle, const char* psPkg, int iPkgLen, STUMsgProperty3 *pMsgProperty);
```

Parameter description:

Parameter	Description
pHandle [in]	Connection handle, the value returned when calling Mr3Init.
psPkg [in]	The message packet buffer to be sent.
iPkgLen [in]	The length of message packet in the buffer.
pMsgPropery [in/out]	Message packet attribute. (1) The input of m_szSourceUserID is ignored, and the output is the user ID of the user set in the access client. (2) The input of m_szSourceAppID is ignored, and the output is the application ID of the application set in Mr2Init. (3) m_szDestUserID and m_szDestAppID input are the destination address of the message packet - the destination user ID and the destination application ID, and the output is unchanged. (4) The m_szPkgID input is a unique message packet ID. This value is either a unique identifier generated by the user calling the MrCreatePkgID function, and the output is unchanged; or the input is empty (ie, '\0'), then the output is the unique message packet ID automatically assigned by the system. For each packet sent, the m_szPkgID should be unique across the entire financial data exchange platform, and different packages should use different m_szPkgIDs. Although the message ID generated by the Mr2CreatePkgID function is unique, different users may take different calling methods because of different programming methods, some may not call the message by standard methods, or use it for special purposes by adding some extra information, even

Parameter	Description
	resulting in programming errors, etc., thus m_szPkgID or m_szCorrPkgID may be repeated. Users therefore cannot simply use m_szPkgID or m_szCorrPkgID as the globally unique ID. FDEP will only correctly transmit the packet to the destination, and the other party should, after receiving the message packet, perform different business processes depending on different situations, e.g. checking user serial number, fund account number, user name and other information, and determining the package related information by checking the business information. (5) The m_szCorrPkgID, m_szUserData1, and m_szUserData2 inputs are user-defined data, and the output is unchanged. (6) The m_ucFlag input is the required packet processing bit flag, and the output is unchanged. (7) m_ucBizType input refers to the business type of the data. (8) m_ucPriority input refers to the priority level of the message transmission. (9) m_ucSensitiveLevel input refers to the sensitivity level of the message content. (10) The m_szMsgType input refers to the type of the message.
iMillSecTimeo [in]	The maximum timeout for sending in milliseconds.

Description of return value:

Return value	Description
0	Successful
Others	Failed

4.4.12 Mr3ReceiveTopicMsg

Topic message packet receiver function. This function has functions similar function to the Mr3Receive2 function, but the user has to allocate

Note: When receiving a message packet, either the callback function OnReceive in the function Mr3Init, or Mr3Receive1/Mr3Receive1_FreeBuf, Mr3Receive2 or Mr3Receive3/Mr3Receive1_FreeBuf can be used for reception purpose, but not both.

Function prototype:

```
int _stdcall    Mr3ReceiveTopicMsg(void* pHandle, char* psPkg, int* piOutPkgLen,
int iBufLenIn, STUMsgProperty3 *pMsgProperty, int iMillSecTimeo);
```

Parameter description:

Parameter	Description
pHandle [in]	Connection handle, the value returned when calling Mr3Init.
psPkg [out]	Message packet receive buffer.
piOutPkgLen [out]	The actual length of the message packet received.
iBufLenIn [in]	Size of message packet buffer.
pMsgPropery [in/out]	The input will be considered as the receiving condition, and six fields in the structure are used as the judgment conditions: m_szSourceUserID, m_szSourceAppID, m_szPkgID, m_szCorrPkgID, m_szUserData1, and m_szUserData2. Other fields are ignored. If one of the input values in the six fields is empty, the field is also excluded from the judgment condition, that is, the actual judgment condition is a non-empty field among the six fields. The criteria for compliance are that all corresponding fields in the message attribute are the same as these non-empty condition field values. If the 6 fields are all empty, the first message packet in the queue is received. Note that there are special treatments for the input condition of m_szCorrPkgID: when the value of the m_szCorrPkgID string is empty, it can match the message with the m_szCorrPkgID field as an arbitrary value; when the value of the m_szCorrPkgID string is "<EMPTY>", then it only matches the message with m_szCorrPkgID field being empty; when the value of the m_szCorrPkgID string is "<NOEMPTY>", it only matches the message with m_szCorrPkgID field being non-empty. The output is the attribute of the packet received.
iMillSecTimeo [in]	The maximum timeout for receiving in milliseconds. This time is the maximum time interval between sending packet request and receiving or not receiving a response by the function for internal control purpose, rather than the actual time interval at which the function actually executes. For example, if there is no eligible packet at this time, even if the value is 1 minute, the function will return the packet immediately as soon as the response is received, unnecessary to wait until 1 minute before returning. This value is related to the busy program of the system. If the system is busy, the value can be appropriately larger. If the system is idle, the value may be appropriately smaller. In general, the value is recommended to be 2,000, namely 2 seconds.

Description of return value:

Return value	Description
0	Successful
Others	Failed

4.4.13 Mr3SendFile

Function prototype:

```
int _stdcall Mr3SendFile(void *pHandle, char *psFileName, char *pReserveInfo, int iReserveInfoLen, char *psFileType, STUMsgProperty3 *parrMsgProperty3, int iArrayLen);
```

Parameter description:

Parameter	Description
pHandle [in]	The handle returned by the Mr3Init function, which is used as a parameter to be called by other functions.
psFileName [in]	The name of the file to be sent, with maximum length of 496 bytes.
pReserveInfo [in]	File reserved information, up to 48KB in length.
iReserveInfoLen [in]	Length of the file reservation information buffer.
psFileType [in]	For the file type, the receiver will create a corresponding folder in its receiving directory to store the received file, with a maximum length of 63.
parrMsgProperty3 [in]	Property parameter structure array, support group sending.
iArrayLen [in]	The length of the Property parameter structure array, which supports group sending, and the value must be greater than 0.

Description of return value:

Return value	Description
0	Successful
Others	Failed

Special notes:

When sending a file, if the file task FileName, STUProperty structure information and the same task of FileType field sent this time is already in the send queue, the task addition will fail, and the system log will prompt that it is not allowed to be added.

4.4.14 Mr3ReceiveFile

Function prototype:

```
int _stdcall Mr3ReceiveFile(void *pHandle, char *psSrcFileName, int *piSrcFileNameOutLen, char *psDestFileName, int *piDestFileNameOutLen, char *pFileReserveInfo, int *piReserveInfoOutLen, STUMsgProperty3 *pMsgProperty3);
```

Parameter description:

Parameter	Description
pHandle [in]	The handle returned by the Mr3Init function, which is used as a parameter to be called by other functions.
psSrcFileName [out]	The file path name of the sender, it is recommended to develop a large enough memory space for storage in advance.
piSrcFileNameOutLen [out]	Length of file path name of the sender
psDestFileName [out]	The destination file name of the receiver, it is recommended to develop a large enough memory space for storage in advance.
piDestFileNameOutLen [out]	Length of destination file name of the receiver
pFileReserveInfo [out]	File reserve information, it is recommended to develop a large enough memory space for storage in advance, and the sender supports sending up to 48KB.
piReserveInfoOutLen [out]	Length of file reserve information
parrMsgProperty3 [out]	Property parameter structure array

Description of return value:

Return value	Description
0	Successful
Others	Failed

4.4.15 Mr3Destroy

Resource release function. Release all resources after the FDEAPI is used. This function is usually the last FDEAPI function to be called.

Function prototype:

```
void _stdcall   Mr3Destroy(void* pHandle);
```

Parameter description:

Parameter	Description
pHandle [in]	Connection handle, the value returned when calling Mr3Init.

Description of return value:

Return value	Description
No	

4.4.16 Mr3GetVersion

Get the version number of the API. This function can be called any time after the dynamic library is loaded.

Function prototype:

```
void _stdcall Mr3GetVersion(char* psBufVersion, int iBufLen);
```

Parameter description:

Parameter	Description
psBufVersion [out]	Returned character string of the API version number. The format of the version number is "01.04.20190630", where 01 indicates a large version number, 04 indicates a small version number, and 20190630 the date the version was released.
iBufLen[in]	It indicates the length of the psBufVersion buffer, which should be greater than or equal to 15.

Description of return value:

Return value	Description
No	

4.4.17 Mr3RegRecvCondition

Register push-down condition function. The so-called registration push-down condition means that, according to the normal function calling rule, the calling receiver function sends a plurality of receiving conditions from the FDEAPI to the FDMR, and then the FDMR sends a packet satisfying the condition to the FDEAPI. If the registered push-down function is called, the FDMR will actively push down the qualified packets to the FDEAPI, which will increase the single link transmission rate of FDMR to FDEAPI at the cost of flexibility to a certain extent. This function is only called when there is a high requirement for single-link traffic. The push-down condition finally called is valid and the previous conditions will be cleared when called more than once.

Function prototype:

```
int _stdcall Mr3RegRecvCondition(void* pHandle, STUMsgProperty3* pArrMsgPropery, int iArrayCount);
```

Parameter description:

Parameter	Description
pHandle [in]	Connection handle, the value returned when calling Mr3Init.
pArrMsgPropery [in/out]	The input will be considered as the receiving condition, and six fields in the structure are used as the judgment conditions: m_szSourceUserID, m_szSourceAppID, m_szPkgID, m_szCorrPkgID, m_szUserData1, and m_szUserData2. Other fields are ignored. If one of the input values in the six fields is empty, the field is also excluded from the judgment condition, that is, the actual judgment condition is a non-empty field among the six fields. The criteria for compliance are that all corresponding fields in the message attribute are the same as these non-empty condition field values. If the 6 fields are all empty, the first message packet in the queue is received. Note that there are special treatments for the input condition of m_szCorrPkgID: when the value of the m_szCorrPkgID string is empty, it can match the message with the m_szCorrPkgID field as an arbitrary value; when the value of the m_szCorrPkgID string is "<EMPTY>", then it only matches the message with m_szCorrPkgID field being empty; when the value of the m_szCorrPkgID string is "<NOEMPTY>", it only matches the message with m_szCorrPkgID field being non-empty. The output is the attribute of the packet received.
iArrayCount [in]	The number of conditions received, that is, the number of elements in the array of message attribute structures.

Description of return value:

Return value	Description
0	Successful
Others	Failed

4.4.18 Mr3GetPeerUserStat

Get the status of the peer communication user. This function can be used to know the status of peer users after a period of successful initialization by calling Mr3Init.

Note: The status information obtained on peer users is not real-time, may be subject to certain delay.

Function prototype:

```
int _stdcall Mr3GetPeerUserStat(void* pHandle, const char* psPeerUserID, int* piOutStat);
```

Parameter description:

Parameter	Description
pHandle [in]	Connection handle, the value returned when calling Mr3Init.
pPeerUserID [in]	Communication peer user ID
piOutStat [out]	Return the current status information of peer user: Return value: 1-Online; 0-Offline

Description of return value:

Return value	Description
0	Successful
Others	Failed, detailed in Section 4.1.4.

4.4.19 Mr3CancelSendFile

Cancel sending files

Function prototype:

```
int _stdcall Mr3CancelSendFile(void *pHandle, char *psFileName, char *pFileResverInfo, int iReserveInfoLen, STUMsgProperty3 *parrMsgProperty3, int iArrayLen);
```

Parameter description:

Parameter	Description
pHandle [in]	The handle returned by the Mr3Init function, which is used as a parameter to be called by other functions.
psFileName [in]	File path name
pFileReserveInfo [in]	File reserve information.
piReserveInfoOutLen [in]	Length of file reserve information.
parrMsgProperty3 [in]	Property parameter structure array
iArrayLen [in]	Length of property parameter structure array, the value must be greater than 0.

Description of return value:

Return value	Description
0	Successful
Others	Failed

4.4.20 Mr3GetSendFileStatus

Get the status of the send file.

Function prototype:

```
int _stdcall Mr3GetSendFileStatus(void *pHandle, char *psFileName, char *pFileResverInfo, int iReserveInfoLen, STUMsgProperty3 *pMsgProperty3, int *piFileStatus, unsigned int *puiPercent, int *piFileThoughOutKbps);
```

Parameter description:

Parameter	Description
pHandle [in]	The handle returned by the Mr3Init function, which is used as a parameter to be called by other functions.
psFileName [in]	File path name
pFileReserveInfo [in]	File reserve information.
piReserveInfoOutLen [in]	Length of file reserve information.
parrMsgProperty [in]	Property parameter structure
piFileStatus [out]	File task status, specific number should be based on the macro definition.
puiPercent [out]	Percentage of the task being transferred, 0%-100%.
piFileThoughOutKbps [out]	Transmission rate of the task, in kbps.

Description of return value:

Return value	Description
0	Successful
Others	Failed

4.4.21 Mr3GetAppStatus

Get status information of an app instance.

Function prototype:

```
int _stdcall Mr3GetAppStatus(void* pHandle, const char* psAppID, STUAppStatus*
pArrAppStatus, int* iArrLen);
```

Parameter description:

Parameter	Description
pHandle [in]	The handle returned by the Mr3Init function, which is used as an argument to other function calls.
psAppID[in]	If you fill in psAppID, you will query the status information of a single app; if you do not fill in, you will query all the apps.
pArrAppStatus[in/out]	pArrAppStatus returns an array of status information.
iArrLen[in/out]	iArrLen returns the length of the status information array.

Description of return value:

Return value	Description
0	Successful
Others	Failed

4.4.22 Calling sequence

The following figure is not a flowchart, but the calling sequence of main functions in FDEAPI. To use FDEAPI, you must first call the function Mr3Init, and finally Mr3Destroy, accompanied by the calling of message sent and received.

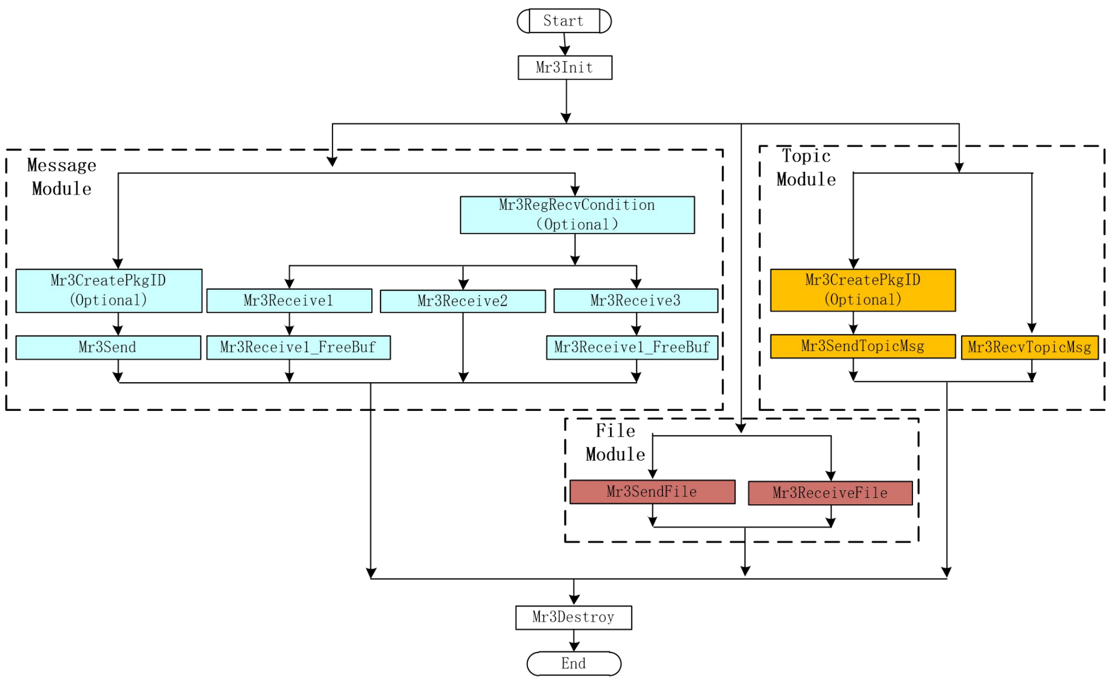


Fig. 6 Calling sequence of main functions of FDEAPI

Note: When receiving a message packet, either the callback function OnReceive in the function Mr3Init, or Mr3Receive1/Mr3Receive1_FreeBuf, Mr3Receive2 or Mr3Receive3/Mr3Receive1_FreeBuf can be used for reception purpose, but not both.

5 Examples of C programming

Create a console application project, add the platform.h, inifile.h, inifile.cpp, loadmrapi.h, loadmrapi.cpp, mrapi.h, mr3api.h, demo.cpp file in the release package rar to the project, then compile it. Only the demo.cpp code is attached below.

```
#include "loadmrapi.h"
#include "inifile.h"
#include <stdio.h>
#include <string>
#include <string.h>
#if (G_PLATFORM_OS==G_PLATFORM_OS_WINDOWS)
#include <Windows.h>
#include <WinBase.h>
#include <process.h>
#else
```

```
#include <pthread.h>
#endif
#include <time.h>

using namespace std;

#if (G_PLATFORM_OS==G_PLATFORM_OS_WINDOWS)
#define LIB_MRAPI_FILE  "./mrapi.dll"
#define THREAD_RTN unsigned __stdcall
#else
#define LIB_MRAPI_FILE  "./libmrapi.so"
#define THREAD_RTN void*
#define   Sleep(waitTime) usleep((waitTime)*1000)
#endif

typedef struct _tagCfgParam
{
	string m_ssAppID;
	string m_ssAppPasswd;
	STUConnInfo3 m_oIpAddress[2];		//mr addresses。
	STUMsgProperty3 m_oSendMsgProperty;	//attribute。
} STUCfgParam;

bool LoadCfg(STUCfgParam* pCfgParam)
{
	//config file name
	string ssCfgFile = "./demo.ini";

	char szAppPasswd[10];
	memset(szAppPasswd, 0, 10);
	memset(&pCfgParam->m_oIpAddress, 0, 2*sizeof(pCfgParam->m_oIpAddress));
	memset(&pCfgParam->m_oSendMsgProperty, 0,
sizeof(pCfgParam->m_oSendMsgProperty));

	read_profile_string("Config", "AppID",
pCfgParam->m_oSendMsgProperty.m_szSourceAppID,
sizeof(pCfgParam->m_oSendMsgProperty.m_szSourceAppID), "app1",
ssCfgFile.c_str());
	read_profile_string("Config", "AppPasswd", szAppPasswd, sizeof(szAppPasswd),
"1", ssCfgFile.c_str());
	read_profile_string("Config", "MrIP1", pCfgParam->m_oIpAddress[0].m_szMRIP,
sizeof(pCfgParam->m_oIpAddress[0].m_szMRIP), "127.0.0.1",ssCfgFile.c_str());
	pCfgParam->m_oIpAddress[0].m_usMRPort = (unsigned
short)read_profile_int("Config", "MrPort1", 0, ssCfgFile.c_str());
```

```
    read_profile_string("Config", "MrIP1", pCfgParam->m_oIpAddress[1].m_szMRIP,
sizeof(pCfgParam->m_oIpAddress[1].m_szMRIP), "127.0.0.1", ssCfgFile.c_str());
    pCfgParam->m_oIpAddress[1].m_usMRPort = (unsigned
short)read_profile_int("Config", "MrPort2", 0, ssCfgFile.c_str());

    pCfgParam->m_ssAppID =
pCfgParam->m_oSendMsgProperty.m_szSourceAppID;
    pCfgParam->m_ssAppPasswd = szAppPasswd;

    read_profile_string("Config", "SourceUserID",
pCfgParam->m_oSendMsgProperty.m_szSourceUserID,
sizeof(pCfgParam->m_oSendMsgProperty.m_szSourceUserID), "", ssCfgFile.c_str());
    read_profile_string("Config", "DestUserID",
pCfgParam->m_oSendMsgProperty.m_szDestUserID,
sizeof(pCfgParam->m_oSendMsgProperty.m_szDestUserID), "",ssCfgFile.c_str());
    read_profile_string("Config", "DestAppID",
pCfgParam->m_oSendMsgProperty.m_szDestAppID,
sizeof(pCfgParam->m_oSendMsgProperty.m_szDestAppID), "", ssCfgFile.c_str());
    read_profile_string("Config", "UserData1",
pCfgParam->m_oSendMsgProperty.m_szUserData1,
sizeof(pCfgParam->m_oSendMsgProperty.m_szUserData1), "",ssCfgFile.c_str());
    read_profile_string("Config", "UserData2",
pCfgParam->m_oSendMsgProperty.m_szUserData2,
sizeof(pCfgParam->m_oSendMsgProperty.m_szUserData2), "",ssCfgFile.c_str());
    pCfgParam->m_oSendMsgProperty.m_ucFlag = (unsigned
char)read_profile_int("Config", "Flag", 0, ssCfgFile.c_str());
    pCfgParam->m_oSendMsgProperty.m_ucBizType = (unsigned
char)read_profile_int("Config", "BizType", 0, ssCfgFile.c_str());
    pCfgParam->m_oSendMsgProperty.m_ucPriority = (unsigned
char)read_profile_int("Config", "Priority", 0, ssCfgFile.c_str());
    pCfgParam->m_oSendMsgProperty.m_ucSensitiveLevel = (unsigned
char)read_profile_int("Config", "SensitiveLevel", 0, ssCfgFile.c_str());

    return true;
}

void MySleep(int iMillSecond)
{
    Sleep(iMillSecond);
}


void*   g_pHandle = NULL;
```

```
int OnReceive(const char* psPkg, int iPkgLen, const STUMsgProperty3* pMsgPropery,
void* pvUserData)
{
    static int s_iRecvCount;
    ++s_iRecvCount;

    printf("recv ok: PkgContent[%s]\n, pkgID[%s], src[%s.%s], dest[%s.%s],
pkgLen[%d], CorrPkgID[%s], UserData1[%s], UserData2[%s] \n", psPkg,
pMsgPropery->m_szPkgID, pMsgPropery->m_szSourceUserID,
pMsgPropery->m_szSourceAppID, pMsgPropery->m_szDestUserID,
pMsgPropery->m_szDestAppID, iPkgLen, pMsgPropery->m_szCorrPkgID,
pMsgPropery->m_szUserData1, pMsgPropery->m_szUserData2);
    printf("client recv count[%d]\n", s_iRecvCount);

    if (pMsgPropery->m_szCorrPkgID[0]=='\0')
    {
        //if m_szCorrPkgID is empty, then it is a request; otherwise, it is a response.

        STUMsgProperty3 oMsgProperySend;
        memset(&oMsgProperySend, '\0', sizeof(STUMsgProperty3));
        strcpy(oMsgProperySend.m_szDestUserID,
pMsgPropery->m_szSourceUserID);
        strcpy(oMsgProperySend.m_szDestAppID,
pMsgPropery->m_szSourceAppID);
        strcpy(oMsgProperySend.m_szCorrPkgID, pMsgPropery->m_szPkgID); //set
response's CorrPkgID = PkgID of request pkg.
        strcpy(oMsgProperySend.m_szUserData1, pMsgPropery->m_szUserData1);
        strcpy(oMsgProperySend.m_szUserData2, pMsgPropery->m_szUserData2);
        oMsgProperySend.m_ucFlag = 0;

        //construct response message。
        const char* psResponsePkg = "Response pkg.";
        int iResponsePkgLen = strlen(psResponsePkg);
        Mr3SendFunc(g_pHandle, psResponsePkg, iResponsePkgLen,
&oMsgProperySend, 2000);
    }
    else
    {
        //if m_szCorrPkgID is empty, then it is a request; otherwise, it is a response.
        //...
    }

    return 0;
```

```
}

//receive thread
THREAD_RTN RecvThrd(void *pvParam)
{
    while (1)
    {
        STUMsgProperty3 oMsgPropery;
        memset(&oMsgPropery, '\0', sizeof(STUMsgProperty3));

        int iRecvPkgLen = 0;
        char* psPkg = NULL;
        int iRecvRet = Mr3Receive1Func(g_pHandle, &psPkg, &iRecvPkgLen,
&oMsgPropery, 1000);
        if(iRecvRet==MR3_ERRCODE_OK)
        {
            static int s_iRecvCount;
            ++s_iRecvCount;
            printf("recv ok: PkgContent[%s]\n, pkgID[%s], src[%s.%s],
dest[%s.%s], pkgLen[%d], CorrPkgID[%s], UserData1[%s], UserData2[%s] \n",
psPkg, oMsgPropery.m_szPkgID, oMsgPropery.m_szSourceUserID,
oMsgPropery.m_szSourceAppID, oMsgPropery.m_szDestUserID,
oMsgPropery.m_szDestAppID, iRecvPkgLen, oMsgPropery.m_szCorrPkgID,
oMsgPropery.m_szUserData1, oMsgPropery.m_szUserData2);
            printf("client recv count[%d]\n", s_iRecvCount);

            if (oMsgPropery.m_szCorrPkgID[0]=='\0')
            {
                //if m_szCorrPkgID is empty, then it is a request; otherwise, it is a
response
                //...

                STUMsgProperty3 oMsgProperySend;
                memset(&oMsgProperySend, '\0', sizeof(STUMsgProperty3));
                strcpy(oMsgProperySend.m_szDestUserID,
oMsgPropery.m_szSourceUserID);
                strcpy(oMsgProperySend.m_szDestAppID,
oMsgPropery.m_szSourceAppID);
                strcpy(oMsgProperySend.m_szCorrPkgID, oMsgPropery.m_szPkgID);
//set response's CorrPkgID = PkgID of request pkg.
                strcpy(oMsgProperySend.m_szUserData1,
oMsgPropery.m_szUserData1);
                strcpy(oMsgProperySend.m_szUserData2,
oMsgPropery.m_szUserData2);
```

```
                    oMsgProperySend.m_ucFlag = 0;

                    //construct response message
                    const char* psResponsePkg = "Response pkg.";
                    int iResponsePkgLen = strlen(psResponsePkg);
                    Mr3SendFunc(g_pHandle, psResponsePkg, iResponsePkgLen,
&oMsgProperySend, 2000);
               }
               else
               {
                    //if m_szCorrPkgID is empty, then it is a request; otherwise, it is a
response
                    //...
               }
               Mr3Receive1_FreeBufFunc(psPkg);
          }
          else
          {
               //errno（referenced by mr3api.h）。
               MySleep(20);
          }
     }

     return 0;
}

//send thread。
THREAD_RTN SendThrd(void *pvParam)
{
     //create send data。
     string ssPkg;
     ssPkg +=" <IFTS Len=\" 1205\" DataVer =\"1.0.0.1\" SeqNo=\"\" Type=\"B\"
Dup=\"N\" CheckSum=\"\">\n";
     ssPkg += "   <MsgText> \n";
     ssPkg += "      <Acmt.001.01> \n";
     ssPkg += "         <MsgHdr> \n";
     ssPkg += "            <Ver>1.0</Ver> \n";
     ssPkg += "            <SysType>0</SysType> \n";
     ssPkg += "            <InstrCd>11002</InstrCd> \n";
     ssPkg += "            <Sender> \n";
     ssPkg += "              <InstType>S</InstType> \n";
     ssPkg += "               <BrchId>1</BrchId> \n";
     ssPkg += "            </Sender> \n";
     ssPkg += "            <Recver> \n";
```

```
    ssPkg += "              <InstType>B</InstType> \n";
    ssPkg += "              <BrchId>2</BrchId> \n";
    ssPkg += "           </Recver> \n";
    ssPkg += "           <Date>20061216</Date> \n";
    ssPkg += "           <Time>114340</Time> \n";
    ssPkg += "           <Ref> \n";
    ssPkg += "            <IssrType>S</IssrType> \n";
    ssPkg += "            <Ref>9</Ref> \n";
    ssPkg += "           </Ref> \n";
    ssPkg += "         </MsgHdr> \n";
    ssPkg += "         <Cust> \n";
    ssPkg += "           <Name>郭达</Name> \n";
    ssPkg += "           <CertType>10</CertType> \n";
    ssPkg += "           <CertId>42218901</CertId> \n";
    ssPkg += "           <Type>INVE</Type> \n";
    ssPkg += "           <Sex>M</Sex> \n";
    ssPkg += "           <Ntnl>CHN</Ntnl> \n";
    ssPkg += "           <Addr>八卦四路22号</Addr>\n";
    ssPkg += "           <PstCd>518029</PstCd> \n";
    ssPkg += "           <Email></Email> \n";
    ssPkg += "           <Fax></Fax> \n";
    ssPkg += "           <Mobile>13843453</Mobile> \n";
    ssPkg += "           <Tel></Tel> \n";
    ssPkg += "          </Cust> \n";
    ssPkg += "          <Agt> </Agt>\n";
    ssPkg += "          < BkAcct> </ BkAcct>\n";
    ssPkg += "          < ScAcct> </ ScAcct>\n";
    ssPkg += "          < ScAcct> </ ScAcct>\n";
    ssPkg += "        </Acmt.001.01>\n";
    ssPkg += "    </MsgText>\n";
    ssPkg += " </IFTS>\n";

    STUCfgParam* pCfgParam = (STUCfgParam*)pvParam;

    //send every 1 second。
    while (1)
    {
        STUMsgProperty3 oMsgPropery;
        memcpy(&oMsgPropery, &(pCfgParam->m_oSendMsgProperty),
sizeof(STUMsgProperty3));

        int iSendRet = Mr3SendFunc(g_pHandle, ssPkg.c_str(), (int)ssPkg.length(),
&oMsgPropery, 2000);
        if (iSendRet == MR3_ERRCODE_OK)
        {
            //success
            printf("send ok: PkgID[%s], src[%s.%s], dest[%s.%s], pkgLen[%d], 
CorrPkgID[%s], UserData1[%s], UserData2[%s] \n", oMsgPropery.m_szPkgID, 
oMsgPropery.m_szSourceUserID, oMsgPropery.m_szSourceAppID, 
oMsgPropery.m_szDestUserID, oMsgPropery.m_szDestAppID, (int)ssPkg.length(), 
oMsgPropery.m_szCorrPkgID, oMsgPropery.m_szUserData1, 
oMsgPropery.m_szUserData2);
        }
        else
        {
            //fail
            printf("send failed[%d]: PkgID[%s], src[%s.%s], dest[%s.%s], 
pkgLen[%d], CorrPkgID[%s], UserData1[%s], UserData2[%s] \n", iSendRet, 
oMsgPropery.m_szPkgID, oMsgPropery.m_szSourceUserID, 
oMsgPropery.m_szSourceAppID, oMsgPropery.m_szDestUserID, 
oMsgPropery.m_szDestAppID, (int)ssPkg.length(), oMsgPropery.m_szCorrPkgID, 
oMsgPropery.m_szUserData1, oMsgPropery.m_szUserData2);
        }

        MySleep(1000);
    }

    return 0;
}


int main(int argc, char* argv[])
{
    //load mrapi.dll
    if (0 != loadmrapi(LIB_MRAPI_FILE))
    {
        printf("load %s error.\n", LIB_MRAPI_FILE);
        return -1;
    }

    //load config
    STUCfgParam oCfgParam;
    LoadCfg(&oCfgParam);

    const char* psUserData = "UserData";

    //initialize
    g_pHandle = Mr3InitFunc(oCfgParam.m_ssAppID.c_str(),
```

深圳证券通信有限公司
SHENZHEN SECURITIES COMMUNICATION CO.,LTD.

```
oCfgParam.m_ssAppPasswd.c_str(), NULL, oCfgParam.m_oIpAddress, 2,
(void*)psUserData);
     if(g_pHandle==NULL)
     {
          //fail
          unloadmrapi();
          printf("ERROR: Mr3Init failed, exit.\n");
          return -1;
     }
     else
     {
          //success
          printf("Mr3Init OK.\n");
     }

#if (G_PLATFORM_OS==G_PLATFORM_OS_WINDOWS)

     //start receive threads
     if(_beginthreadex(NULL, 0,  RecvThrd, &oCfgParam, 0, NULL)==0)
     {
          unloadmrapi();
         printf("ERROR: create recv thread failed, exit.\n");
         return -1;
     }
     else
     {
          printf("create recv thread OK.\n");
     }

     //start send threads
     if(_beginthreadex(NULL, 0,  SendThrd, &oCfgParam, 0, NULL)==0)
     {
          unloadmrapi();
         printf("ERROR: create send thread failed, exit.\n");
         return -1;
     }
     else
     {
          printf("create send thread OK.\n");
     }

#else
     //start receive threads
     pthread_t recvpid;
```

```
    if(pthread_create(&recvpid, NULL,  RecvThrd, &oCfgParam)!=0)
    {
         unloadmrapi();
        printf("ERROR: create recv thread failed\n");
        return -1;
    }
    else
    {
         printf("create recv thread OK.\n");
    }

    //start send threads
    pthread_t sendpid;
    if(pthread_create(&sendpid, NULL,  SendThrd, &oCfgParam)!=0)
    {
         unloadmrapi();
        printf("ERROR: create send thread failed\n");
        return -1;
    }
    else
    {
         printf("create send thread OK.\n");
    }
#endif

    while (1)
    {
         //check link status every 1 seconds。
         if (Mr3IsLinkOKFunc(g_pHandle)==0)
         {
              printf("WARN: Link not ok.\n");
         }

         MySleep(1000);
    }

    //release resource
    Mr3DestroyFunc(g_pHandle);
    unloadmrapi();

    return 0;
}
```

6 Java interface description

6.1.1 Function list

There are 17 mrapi class methods (Chapter 7 of this manual has specific instructions for creating mrapi classes), which correspond to the 17 functions mentioned in 4.4.1 of this article, as follows:

No.	Function name	Functional features
1	Mr3Init	Initialize, obtain related resources, and try to establish a connection with the access client FDAP.
2	Mr3IsLinkOK	Check and judge whether the current connection with the access client FDAP is normal.
3	Mr3CreatePkgID	Generate a message package ID.
4	Mr3Send	Send a single message to the message hub through the FDAP, requesting forwarding.
5	Mr3Receive1	Receive the message forwarded by the message hub in the manner 1 condition.
6	Mr3Receive3	Receive the message forwarded by the message hub in the condition 3 condition.
7	Mr3Destroy	Disconnect from the FDAP and release related resources.
8	Mr3GetVersion	Get the version number of the API.
9	Mr3RegRecvCondition	The registration package pushes down the conditions, pushing all conditions at once.
10	Mr3GetPeerUserStat	Get the status of the communication peer user.
11	Mr3SendTopicMsg	A subject message is sent to the message center through the FDAP, and the request is forwarded.
12	Mr3ReceiveTopicMsg	After subscribing to the topic, receive the topic message forwarded by the hub.
13	Mr3SendFile	Send a single file message to the message center through the FDAP, requesting forwarding.
14	Mr3ReceiveFile	Receive file messages forwarded by the hub.
15	Mr3CancelSendFile	Cancel sending the file.
16	Mr3GetSendFileStatus	Get the status of the sent file.
17	Mr3GetAppStatus	Gets the status information of the app instance.

Note: The MRAPI dynamic link library also contains functions beginning with MR and

MR2, which are reserved for compatibility with the old version of the interface; functions beginning with MR3 are new interfaces, and old and new interface functions cannot be mixed.

6.1.2 Mr3Init

Initialization method when connecting to FDAP. This method initializes FDEAPI, allocates relevant resources, and attempts to establish a communication connection with the access client.

When FDEAPI calls this initialization method, it will try to connect to the client using TCP. Generally, the connection will be successful before the method returns. If the access client is temporarily unavailable or there is a network problem, when you call this method to return, you may not be connected to the access client. Because FDEAPI has the function of disconnection and reconnection, the API will automatically retry with the other party connection.

Function prototype:

```
public native static int Mr3Init(String sAppId, int iInstanceId, String sAppPwd, String
sIp, short uPort, String sIpbak, short uPortbak);
```

Parameter description:

Parameter	Description
sAppId[in]	The application ID of this application.
iInstanceId[in]	The instance ID of this application.
sAppPwd[in]	The password set by the application on the access client. The password must match the preset to continue.
sIp[in]	The IP address of the first access client message router.
uPort[in]	The first connection port to the client message router.
sIpbak[in]	The IP address of the second access client message router.
uPortbak[in]	Connection port for the second access client message router.

Note: Mr3Init will first try to connect to the designated first message router. If unsuccessful, try to connect to the second message router. When using FDEAPI, as long as you specify to connect to one of the message routers, load balancing is automatically performed among all message routers that access the client. In other words, the message router to which the application finally connects may not be the two specified by the parameters.

Description of return value:

Return value	Description
0	initialization failed.
Non-0	initialized successfully. (Must be judged in conjunction with Mr3IsLinkOK)

6.1.3 Mr3IsLinkOK

Connection status judgment method. This method determines whether the current connection with the access client is normal. Because the message exchange of the entire financial data exchange platform is asynchronous, and FDEAPI has the function of automatic re-connection, even if the current connection is abnormal, it does not mean that the system cannot work normally. After the connection is automatically restored, various tasks will continue.

Note: After Mr3Init obtains the return value, it does not mean that the network has been successfully connected, and the network connection may be in progress. Therefore, after using the Mr3Init method, you should wait for a few seconds before performing the Mr3IsLinkOK judgment. If Mr3IsLinkOK judgment is performed immediately after using the Mr3Init method, it may be considered that there is no normal connection.

Function prototype:

```
public native static int Mr3IsLinkOK(String sAppId, int iInstanceId);
```

Parameter description:

Parameter	Description
sAppId[in]	Application ID for this application.
iInstanceId[in]	Instance ID for this application.

Description of return value:

Return value	Description
0	The connection is abnormal.
1	The connection is normal.

6.1.4 Mr3CreatePkgID

Message packet identification generation method. This method generates a globally unique UUID in the entire FDEP, which can be used as the message packet identifier.

Function prototype:

```
public native static byte[]    Mr3CreatePkgID(String sAppId, int iInstanceId);
```

Parameter description:

Parameter	Description
sAppId[in]	Application ID for this application.
iInstanceId[in]	Instance ID for this application.

Description of return value:

Return value	Description
UUID	Generated packet ID.
String with "NULL"	failure.

6.1.5 Mr3Send

Single shot message packet method. This method performs the function of sending a message to a single destination user.

Promise:

If the user calls the Mr3Send method to send a business response packet, the CorrPkgID parameter is uniformly agreed and filled with the PkgID value in the corresponding request packet, so that the requester can find the corresponding request packet according to this field after receiving the response packet. For service request packets or other types of packets, this field is blank.

Function prototype:

```
public native static String Mr3Send( byte[] psPkg, String sSourceUserID,String
sSourceAppID, int iSrcInstanceId, String sDestUserID, String sDestAppID,String
sPkgID, String sCorrPkgID, String sUserData1, String sUserData2, String sMsgType,
byte cFlag, int iBizType, byte cPriority, byte cSensitiveLevel, int iMillSecTimeo);
```

Parameter description:

Parameter	Description
psPkg [in]	Data to be sent.
sSourceUserID [in]	Source user ID.
sSourceAppID [in]	Source application ID.
iSrcInstanceId[in]	The instance ID of this application.
sDestUserID [in]	Destination user ID.
sDestAppID [in]	Purpose application identification.
sPkgID [in]	The packet identifier of the message packet (can be generated by the Mr3CreatePkgID method). This parameter is a unique message packet identifier. The unique identifier that can be generated by the user by calling the Mr3CreatePkgID method, the output is unchanged; if the input is empty, the system will automatically assign a unique message packet identifier. For each packet sent, the PkgID should be unique in the entire financial data exchange platform, and different packets need to use different PkgIDs. Although the message identifier generated by the Mr3CreatePkgID method is unique, due to the different calling methods of different users, the programming method is also different. Some users may call them irregularly, and some users may use them for special purposes, adding some additional Information, and even programming errors may cause PkgID or CorrPkgID to be duplicated. Therefore, users cannot simply use PkgID or CorrPkgID as a globally unique identifier. FDEP is only responsible for correctly transmitting the packet to the destination, and the other party receives the After the message package, different situations should be distinguished for different business processing, such as checking the user's serial number, fund account number, user name and other information, and determining the relevant information of the package by checking the business information.
sCorrPkgID [in]	Related package ID.
sUserData1 [in]	User data 1.
sUserData2 [in]	User data 2.
sMsgType [in]	Message type identification.
cFlag [in]	Message flag, consisting of 8 binary bits. The meaning of each bit is as follows: Bit 0-a value of 1 indicates a persistent message, which needs to be transmitted reliably and is not supported for the time being;

	Bit 1-a value of 1 indicates that the message needs to be compressed for transmission; Bit 2-A value of 1 indicates that the response packet is automatically pushed to the sender.
iBizType [in]	Business Type Flag:0～255。
cPriority [in]	Priority marks: 3 to 5, 5 is the lowest, and 3 is the highest.
cSensitiveLevel [in]	Sensitivity level flag: 0 to 255, with 0 being the lowest and 255 being the highest.
iMillSecTimeo [in]	The maximum timeout for sending in milliseconds. The recommended value is 2000, which is 2 seconds.

Description of return value:

Return value	Description
PkgID	success.
""	failure.

6.1.6 Mr3Receive1

Message packet receiving method 1. Receive the first eligible message from the receiving queue corresponding to this application according to the given conditions, and remove the message from the queue. The memory occupied by the message packets received by this method is allocated internally by FDEAPI.

Function prototype:

```
public native static byte[] Mr3Receive1(String sAppId, int iInstanceId,String sSourceUserID,String sSourceAppID, String sDestUserID, String sDestAppID,String sPkgID, String sCorrPkgID, String sUserData1, String sUserData2, int iMillSecTimeo);
```

Parameter description:

Parameter	Description
sAppId[in]	The application ID of this application.
iInstanceId[in]	The instance ID of this application.
sSourceUserID [in]	Source user identification criteria.
sSourceAppID [in]	The source applies the identification criteria.
sDestUserID [in]	Destination user identification condition.
sDestAppID [in]	Purpose To apply identification conditions.
sPkgID [in]	The packet identification condition of the message packet.
sCorrPkgID [in]	Related package identification conditions.

sUserData1 [in]	User data 1 condition.
sUserData2 [in]	User data 2 condition.
iMillSecTimeo [in]	The maximum receive timeout in milliseconds. This time is the maximum time interval within the method to control the time between sending and receiving a packet request and receiving a response or no response, not the time interval that the method actually executes. For example, if there is no eligible packet at this time, even if the value is 1 minute, as long as the method receives the response internally, it will return immediately without waiting for 1 minute before returning. This value is related to the busy program of the system. If the system is busy, the value can be appropriately large. If the system is idle, the value can be appropriately small. Generally, it is recommended that the value is 2000, which is 2 seconds.

Note: 6 fields in the method parameters are used as the judgment conditions for receiving: SourceUserID, SourceAppID, PkgID, CorrPkgID, UserData1 and UserData2. If an input value in these 6 fields is empty, the field is also excluded from the judgment condition, that is, the actual judgment condition is a non-null field in the 6 fields. Eligibility criteria are that all corresponding fields in the message properties have the same value as these non-empty condition fields. If all 6 fields are empty, the first message packet in the queue is received.

CorrPkgID input conditions have special processing: when the value of the CorrPkgID string is empty, it can match the message in which the CorrPkgID field is an arbitrary value; when the value of the CorrPkgID string is "<EMPTY>", only the message The message in which the CorrPkgID field is empty; when the value of the CorrPkgID string is "<NOEMPTY>", only the message in which the CorrPkgID field is not empty is matched.

Description of return value:

Return value	Description
pkID(64byte)+CorrpkID(64byte)+sourceUserID(32byte) +sourceAppID(32byte)+sourceIntanceId(16byte)+MsgType(8byte) +Flag(8byte) +BizType(8byte) +Priority(8byte)+SensitiveLevel(8byte) +UserData1(256byte) +UserData2(256byte) +data， the above 12 fields are the same as the corresponding input parameters, and data is the real data	Received successfully.
"NULL"or"NULL,errmsg"	Received failed.

6.1.7 Mr3Receive3

Message packet receiving method 3. This method is similar to the Mr3Receive1 method, but this method can receive system error messages, such as when the target address does not exist, the queue is full, the timeout expires, or a system error. Reason code. The memory occupied by the message packets received by this method is allocated internally by FDEAPI.

Function prototype:

```
public native static byte[] Mr3Receive3(String sAppId, int iInstanceId,String
sSourceUserID,String sSourceAppID, String sDestUserID, String sDestAppID, String
sPkgID, String sCorrPkgID, String sUserData1, String sUserData2, int
iMillSecTimeo);
```

Parameter description:

Parameter	Description
sAppId[in]	The application ID of this application.
iInstanceId[in]	The instance ID of this application.
SourceUserID [in]	Source user identification criteria.
SourceAppID [in]	The source applies the identification criteria.
DestUserID [in]	Destination user identification condition.
DestAppID [in]	Purpose To apply identification conditions.
PkgID [in]	The packet identification condition of the message packet.
CorrPkgID [in]	Related package identification conditions.
UserData1 [in]	User data 1 condition.
UserData2 [in]	User data 2 conditions.
iMillSecTimeo [in]	The maximum receive timeout in milliseconds. This time is the maximum time interval within the method to control the time between sending and receiving a packet request and receiving a response or no response, not the time interval that the method actually executes. For example, if there is no eligible packet at this time, even if the value is 1 minute, as long as the method receives the response internally, it will return immediately without waiting for 1 minute before returning. This value is related to the busy program of the system. If the system is busy, the value can be appropriately large. If the system is idle, the value can be appropriately small. Generally, it is recommended that the value is 2000, which is 2 seconds.

Note: 6 fields in the method parameters are used as the judgment conditions for receiving:

SourceUserID, SourceAppID, PkgID, CorrPkgID, UserData1 and UserData2. If an input value in these 6 fields is empty, the field is also excluded from the judgment condition, that is, the actual judgment condition is a non-null field in the 6 fields. Eligibility criteria are that all corresponding fields in the message properties have the same value as these non-empty condition fields. If all 6 fields are empty, the first message packet in the queue is received.

CorrPkgID input conditions have special processing: when the value of the CorrPkgID string is empty, it can match the message in which the CorrPkgID field is an arbitrary value; when the value of the CorrPkgID string is "<EMPTY>", only the message The message in which the CorrPkgID field is empty; when the value of the CorrPkgID string is "<NOEMPTY>", only the message in which the CorrPkgID field is not empty is matched.

Description of return value:

Return value	Description
errcode(4byte) +pkID(64byte) +CorrpkID(64byte) +sourceUserID(32byte) +sourceAppID(32byte) +sourceIntanceId(16byte)+MsgType(8byte) +Flag(8byte) +BizType(8byte) +Priority(8byte) +SensitiveLevel(8byte)+UserData1(256byte) +UserData2(256byte) +data， 12 fields are the same as the corresponding input parameters, and data is the real data. Where errcode = "0000" is normal reception, if errcode is a non- "0000" string, the error message returned by the system, the detailed error is in the data field	Received successfully. Where errcode = "0000" is normal reception, if errcode is a non- "0000" string, the error message returned by the system, the detailed error is in the data field.
"NULL" or"NULL,errmsg"	Received failed.

6.1.8 Mr3Destroy

Resource release method. After FDEAPI is used, release all resources. This method is usually the last FDEAPI method to be called.

Function prototype:

```
public native static void Mr3Destroy(String sAppId, int iInstanceId);
```

Parameter description:

Parameter	Description
sAppId[in]	The application ID of this application.

iInstanceId[in]	The instance ID of this application.

Description of return value:

Return value	Description
无	

6.1.9 Mr3GetVersion

Get the version number of the API. This method can be called at any time after loading the dynamic library.

Function prototype:

```
public native static byte[]    Mr3GetVersion();
```

Parameter description:

Parameter	Description

Description of return value:

Return value	Description
Version number string	

6.1.10 Mr3RegRecvCondition

Register a push down condition method.

When using the receiving method Mr3Receive1 or Mr3Receive3, the user provides multiple receiving conditions at a time, thereby receiving a message that satisfies the conditions. If the registration push-down condition method is used, the system will actively push down the packets that meet the conditions to the user in turn, which will increase the single-link transmission rate at the expense of flexibility. This method is only called when there is high demand for single link traffic. When this method is used multiple times, the push-down condition of the last call is valid and the previous conditions are cleared.

Function prototype:

```
public native static int    Mr3RegRecvCondition(String sAppId, int iInstanceId,String
sSrcUserId, String sSrcAppId, String sDestUserId, String sDestAppId, String sPkgId,
String sCorrPkgId, String sUserData1,String sUserData2);
```

Parameter description:

Parameter	Description
sAppId[in]	The application ID of this application.
iInstanceId[in]	The instance ID of this application.
szSrcUserId [in]	Source user identification criteria.
szSrcAppId [in]	The source applies the identification criteria.
szDestUserId [in]	Destination user identification condition.
szDestAppId [in]	Purpose To apply identification conditions.
szPkgId [in]	The packet identification condition of the message packet.
szCorrPkgId [in]	Related package identification conditions.
szUserData1 [in]	User data 1 condition.
szUserData2 [in]	User data 2 conditions.

Note: 6 fields in the method parameters are used as the judgment conditions for receiving: szSrcUserID, szSrcAppID, szPkgId, szCorrPkgID, szUserData1, and szUserData2. If an input value in these 6 fields is empty, the field is also excluded from the judgment condition, that is, the actual judgment condition is a non-null field in the 6 fields. Eligibility criteria are that all corresponding fields in the message properties have the same value as these non-empty condition fields. If all 6 fields are empty, the first message packet in the queue is received.

The szCorrPkgID input condition has special processing: when the value of the szCorrPkgID string is empty, it can match the message in which the szCorrPkgID field is an arbitrary value; when the value of the szCorrPkgID string is "<EMPTY>", only the message is matched Messages where the szCorrPkgID field is empty; when the value of the szCorrPkgID string is "<NOEMPTY>", only messages whose szCorrPkgID field is non-empty are matched.

Description of return value:

Return value	Description
0	success.
less than 0	failure.

6.1.11 Mr3GetPeerUserStat

Get the status of the communication peer user. Call the Mr3Init method for a period of time after the initialization is successful. Use this method to obtain the status of the peer user.

Function prototype:

```
public native static int  Mr3GetPeerUserStat(String sAppId, int iInstanceId,String sPeerUserID);
```

Parameter description:

Parameter	Description
sAppId[in]	The application ID of this application.
iInstanceId[in]	The instance ID of this application.
szPeerUserID [in]	Communication peer user ID.

Description of return value:

Return value	Description
−1	The method call failed.
0	The method call was successful, and the peer user is not online.
1	The method call was successful, and the peer user was online.

6.1.12 Mr3SendTopicMsg

This function completes the function of sending a topic message.

Promise:

If the user calls the Mr3SendTopicMsg method to send a topic message packet, there is no need to fill in the information of the peer user.

Function prototype:

```
public native static String Mr3SendTopicMsg(byte[] psPkg, String sSourceUserID,String sSourceAppID, int iSrcInstanceId, String sTopicName,String sPkgID, String sCorrPkgID, String sUserData1, String sUserData2, byte cFlag, byte cPriority, byte cSensitiveLevel);
```

Parameter description:

Parameter	Description
psPkg [in]	Data to be sent.
sSourceUserID [in]	Source user ID.

sSourceAppID [in]	Source application ID.
iSrcInstanceId[in]	The instance ID of this application.
sTopicName[in]	Theme name.
sPkgID [in]	The packet identifier of the message packet (can be generated by the Mr3CreatePkgID method). This parameter is a unique message packet identifier. The unique identifier that can be generated by the user by calling the Mr3CreatePkgID method, the output is unchanged; if the input is empty, the system will automatically assign a unique message packet identifier. For each packet sent, the PkgID should be unique in the entire financial data exchange platform, and different packets need to use different PkgIDs. Although the message identifier generated by the Mr3CreatePkgID method is unique, due to the different calling methods of different users, the programming method is also different. Some users may call them irregularly, and some users may use them for special purposes, adding some additional Information, and even programming errors may cause PkgID or CorrPkgID to be duplicated. Therefore, users cannot simply use PkgID or CorrPkgID as a globally unique identifier. FDEP is only responsible for correctly transmitting the packet to the destination, and the other party receives the After the message package, different situations should be distinguished for different business processing, such as checking the user's serial number, fund account number, user name and other information, and determining the relevant information of the package by checking the business information.
sCorrPkgID [in]	Related package ID.
sUserData1 [in]	User data 1.
sUserData2 [in]	User data 2.
cFlag [in]	Message flag, consisting of 8 binary bits. The meaning of each bit is as follows: Bit 0-a value of 1 indicates a persistent message, which needs to be transmitted reliably and is not supported for the time being; Bit 1-a value of 1 indicates that the message needs to be compressed for transmission; Bit 2-A value of 1 indicates that the response packet is automatically pushed to the sender.
cBizType [in]	Service type flag: 0 to 255.
cPriority [in]	Priority marks: 3 to 5, 5 is the lowest, and 3 is the highest.
cSensitiveLevel [in]	Sensitivity level flag: 0 to 255, with 0 being the lowest and 255 being the highest.

Description of return value:

Return value	Description
PkgID	success.
""	failure.

6.1.13 Mr3ReceiveTopicMsg

Topic message packet receiving function. The user needs to enter the length of the topic message packet, the maximum value is 5M.

Subject messages are pushed down and assigned by the hub, without the need to submit registration conditions.

Function prototype:

```
public native static byte[] Mr3ReceiveTopicMsg(String sAppId, int iInstanceId, int iBufferLenIn, String sSourceUserID,String sSourceAppID, String sTopicName);
```

Parameter description:

Parameter	Description
sAppId [in]	The application ID of this application.
iInstanceId[in]	The instance ID of this application.
iBufferLenIn[in]	Message packet buffer size.
sSourceUserID[in]	Source user identification criteria.
sSourceAppID[in]	The source applies the identification criteria.
sTopicName[in]	Theme name.

Note: Three fields in the method parameters are used as the judgment conditions for receiving: SourceUserID, SourceAppID, TopicName. If one of the three fields is empty, the field is also excluded from the judgment condition, that is, the actual judgment condition is a non-null field among the three fields. Eligibility criteria are that all corresponding fields in the message properties have the same value as these non-empty condition fields. If all three fields are empty, the first message packet in the queue is received.

Description of return value:

Return value	Description

pkID(64byte)+CorrpkID(64byte)+topicName(17byte)+sourceUserID(32byte)+sourceAppID(32byte)+sourceInstanceID(16byte) +flag(8byte)+Priority(8byte)+SensitiveLevel(8byte)+ UserData1(256byte) + UserData2(256byte) +data, 11 fields are the same as those in pMsgPropery in the C interface, and data is the real data;	Received successfully.
"NULL"	Received failed.

6.1.14 Mr3SendFile

Function prototype:

```
public native static String Mr3SendFile(String sFileFullPath, String sReserveInfo,String sSubDir, String sSourceUserID,String sSourceAppID, int iSrcInstanceID, String sDestUserID, String sDestAppID, int iDestInstanceID, String sUserData1, String sUserData2, int iBizType);
```

Parameter description:

Parameter	Description
sFileFullPath[in]	The full path of the file.。
sReserveInfo[in]	File retention information.
sSubDir[in]	File type. The receiver will create a corresponding folder in its receiving directory to store the received files.
SourceUserID [in]	Source user ID.
SourceAppID [in]	Source application ID.
SrcInstanceID[in]	Source instance ID.
DestUserID [in]	Destination user ID.
DestAppID [in]	Purpose application identification.
DestInstanceID[in]	Destination instance ID.
UserData1 [in]	User data 1.
UserData2 [in]	User data 2.
BizType [in]	Service type flag: 0 to 255.

Description of return value:

Return value	Description
taskid	success.
“”	failure.

6.1.15 Mr3ReceiveFile

Receive details of the file. The file landing directory is: configuration item in mr.ini + SubDir + sending date on the sending end; the landing file name is composed of: determined by the time value received (to millisecond level) + file task ID to ensure the received file name only.

Function prototype:

```
public native static byte[] Mr3ReceiveFile(String sAppId, int iInstanceId);
```

Parameter description:

Parameter	Description
sAppId[in]	The application ID of this application.
iInstanceId[in]	The instance ID of this application.

Description of return value:

Return value	Description
taskID(128byte) +srcFileName(512byte) +destFileName(512byte) +sourceUserID(32byte)+sourceAppID(32byte) +sourceIntanceId(16byte) +bizType(8byte) + UserData1(256byte) + UserData2(256byte) + ReserveInfo	Received successfully.
"NULL"or"NULL,errmsg"	Received failed.

6.1.16 Mr3CancelSendFile

Function prototype:

```
public native static int Mr3CancelSendFile(String sAppId, int iInstanceId, String sTaskId);
```

Parameter description:

Parameter	Description
sAppId[in]	The application ID of this application.
iInstanceId[in]	The instance ID of this application.
sTaskId [in]	Task id.

Description of return value:

Return value	Description
0	success.
Others	failure.

6.1.17 Mr3GetSendFileStatus

Function prototype:

```
public native static byte[] Mr3GetSendFileStatus(String sAppId, int iInstanceId,
String sTaskId);
```

Parameter description:

Parameter	Description
sAppId[in]	The application ID of this application.
iInstanceId[in]	The instance ID of this application.
sTaskId [in]	Task id.

Description of return value:

Return value	Description
FileStatus(8byte) +Percent(8byte) +FileThoughOutKbps(8byte)	success.
NULL	failure.

6.1.18 Mr3GetAppStatus

Function prototype:

```
public native static   byte[] Mr3GetAppStatus(String sAppId, int iInstanceId, String
sQueryAppId);
```

Parameter description:

Parameter	Description
sAppId [in]	The application ID of this application.
iInstanceId[in]	The instance ID of this application.
sQueryAppId[in]	The appid to be queried.

Description of return value:

Return value	Description
AppID(32byte) +LinkID(32byte) +InstanceID(16byte) +IP(16byte) +Port(8byte) +MrName(32byte) +LoginTime(256byte) +QueueLength(64byte) +TopicQueueLength(64byte)+FileQueueLength(64byte) +SendPkgCountTotal(64byte) +RecvPkgCountTotal(64byte)	success.
NULL	failure.

Note:

1. Query a single app with only one instance, and the status information returned successfully is shown in the table;

2. If the query is a single app and there is only one instance, there is no vertical bar separated by "|"; if all apps are queried, the status information of each instance of each app is separated by a vertical bar.

6.1.19 Calling sequence

Refer to 4.4.22.

7 Examples of Java programming

7.1 Example description

There are many ways to call mrapi.dll dynamic link library from Java. This manual provides a JNI-based method for reference only. Users can choose other methods such as JNA.

7.2 Specific steps

7.2.1 Build project

Create a project directory, copy the five files mrapi.dll, ini / mr.ini, mrapi.h, mr2api.h, and mr3api.h from the mrapi_05.01_win32_java_bin_20190812.rar package to the project directory, and copy them to the directory Create the com / sscc / fdep directory structure in turn. Create the mrapi.java file in the fdep directory. The code of the mrapi.java file is as follows (the mrapi.java file is already provided in the rar package):

```
package com.sscc.fdep;


public class mrapi
{
static
    {
        System.loadLibrary("mrapi");
    }

public static void main(String[] args)
{

}

/********************************* FDEP V3
*********************************/

/* initialize
 * return: 0-fail; not 0-succeess
 */
public native static int MrInit(String App,String AppPwd, String Ip,short Port,String
Ipbak, short Portbak);



/* send message
 * return:    ""-fail   pkgId-success
 */

public native static String MrSend( byte[] psPkg, String SourceUserID,String
SourceAppID, String DestUserID, String DestAppID,
String PkgID, String CorrPkgID, String UserData1, String UserData2, String
ExpiredAbsTime, byte flag, byte Protocltype,int iMillSecTimeo);




//return "NULL" or “NULL,ErrMsg”-fail     otherwise return PkgID(64byte) +
CorrpkID(64byte) + sourceUserID(64byte) + sourceAppID(64byte)
+destAppID(64byte) + UserData1(256byte) + UserData2(256byte) + data

public native static byte[] MrReceive1(String sAppID,String SourceUserID,String
SourceAppID, String DestUserID, String DestAppID,
String PkgID, String CorrPkgID, String UserData1, String UserData2, String
ExpiredAbsTime, byte flag, byte Protocltype,int iMillSecTimeo);




/*
 * return: 0-link_not_ok; 1-link_ok

 */
public native static int MrIsLinkOK(String sAppID);

/*release resource*/
public native static void MrDestroy(String sAppID);


/********************************* FDEP V4
**********************************/

/* initialize
 * return: 0-fail; not 0-success
 */
public native static int Mr2Init(String App,String AppPwd, String Ip,short Port,String
Ipbak, short Portbak);




/* send message
 * return:     ""-fail    pkgId-success
 */

public native static String Mr2Send( byte[] psPkg, String SourceUserID,String
SourceAppID, String DestUserID, String DestAppID,
```

深圳证券通信有限公司
SHENZHEN SECURITIES COMMUNICATION CO.,LTD.

```
String PkgID, String CorrPkgID, String UserData1, String UserData2, String MsgType,
byte flag, byte BizType, byte Priority, byte SensitiveLevel, int iMillSecTimeo);




//return "NULL" or “NULL，ErrMsg”-fail     otherwise return PkgID(64byte) +
CorrpkID(64byte) + sourceUserID(32byte) + sourceAppID(32byte)
+destAppID(32byte) + UserData1(256byte) + UserData2(256byte) + data

public native static byte[] Mr2Receive1(String sAppID,String SourceUserID,String
SourceAppID, String DestUserID, String DestAppID,
String PkgID, String CorrPkgID, String UserData1, String UserData2, int iMillSecTimeo);




/*
 * return: 0-link_not_ok; 1-link_ok
 */
public native static int Mr2IsLinkOK(String sAppID);

/*release resource*/
public native static void Mr2Destroy(String sAppID);


/* get version
 * return version
 */
public native static byte[]  Mr2GetVersion();


/* create pkgID
 * return success：PkgID；fail：NULL
 */
public native static byte[]  Mr2CreatePkgID(String szHandleAppID);

/*
/ return:  1-online，0-offline other-fail（referenced by mr2api.h）
*/
public native static int  Mr2GetPeerUserStat(String szHandleAppID,String
szPeerUserID);


/*
```

```
 * return: 0：success,<0fail
 */
public native static int  Mr2RegRecvCondition(String szHandleAppId,String
szSrcUserId, String szSrcAppId, String szDestUserId, String szDestAppId,
String szPkgId, String szCorrPkgId, String szUserData1,String szUserData2) ;


///  "NULL" or "NULL，errmsg"-fail   otherwise return errcode(4byte)+pkID(64byte) +
CorrpkID(64byte) + sourceUserID(32byte) + sourceAppID(32byte)
+destUserID(32byte) + destAppID(32byte)+UserData1(256byte) +
UserData2(256byte) + data
///   errcode= "0000"-success，errcode !="0000"-fail；
public native static byte[] Mr2Receive3(String sAppID,String SourceUserID,String
SourceAppID, String DestUserID, String DestAppID,
String PkgID, String CorrPkgID, String UserData1, String UserData2, int iMillSecTimeo);


/********************************** FDEP V5
***********************************/
/*
 * initialize
 * return:  0-fail; not 0-success
 * note：range of iInstanceId:(0,65535)；
 */
public native static int Mr3Init(String sAppId, int iInstanceId, String sAppPwd, String
sIp, short uPort, String sIpbak, short uPortbak);

/*
 * send message
 * return:  ""-fail; pkgId-success
 * note：1、psPkg- data to be sent；
 *        2、parameters SourceAppID and iSrcInstanceId are needed；
 */
public native static String Mr3Send( byte[] psPkg, String sSourceUserID,String
sSourceAppID, int iSrcInstanceId, String sDestUserID, String sDestAppID,
                        String sPkgID, String sCorrPkgID, String sUserData1, String
sUserData2, String sMsgType, byte cFlag, int iBizType, byte cPriority, byte
cSensitiveLevel, int iMillSecTimeo);

/*
 * receive message
 * return: success：pkID(64byte) +CorrpkID(64byte) +sourceUserID(32byte)
+sourceAppID(32byte) +sourceIntanceId(16byte)
 *          +MsgType(8byte) +Flag(8byte) +BizType(8byte) +Priority(8byte)
```

```
+SensitiveLevel(8byte)
 *          +UserData1(256byte) +UserData2(256byte) +data;
 *          fail: "NULL" or "NULL,errmsg";
 * note: parameters sAppId and iInstanceId are needed;
 */
public native static byte[] Mr3Receive1(String sAppId, int iInstanceId,String
sSourceUserID,String sSourceAppID, String sDestUserID, String sDestAppID,
                                          String sPkgID, String sCorrPkgID, String
sUserData1, String sUserData2, int iMillSecTimeo);

/*
 * return: 0-link_not_ok; 1-link_ok
 * note: parameters sAppId and iInstanceId are needed;
*/
public native static int Mr3IsLinkOK(String sAppId, int iInstanceId);

/*
 * release resource
 * return: void
 * note: parameters sAppId and iInstanceId are needed;
*/
public native static void Mr3Destroy(String sAppId, int iInstanceId);

/*
 * get version
 * return: version
*/
public native static byte[]  Mr3GetVersion();

/*
 * create pkgID
 * return fail: NULL,errmsg,  success: PkgID
 * note: parameters sAppId and iInstanceId are needed;
*/
public native static byte[]  Mr3CreatePkgID(String sAppId, int iInstanceId);

/*
 * return:  1-online, 0-offline other-fail (referenced by mr3api.h)
 * note: parameters sAppId and iInstanceId are needed;
*/
public native static int  Mr3GetPeerUserStat(String sAppId, int iInstanceId,String
sPeerUserID);

/*
```

```
 * return: 0：success,<0 fail
 * note：1、parameters sAppId and iInstanceId are needed；
*/
public native static int  Mr3RegRecvCondition(String sAppId, int iInstanceId,String
sSrcUserId, String sSrcAppId, String sDestUserId, String sDestAppId,
                                              String sPkgId, String sCorrPkgId,
String sUserData1,String sUserData2);

/*
 * receive message
 * return:  errcode(4byte) +pkID(64byte) +CorrpkID(64byte) +sourceUserID(32byte)
+sourceAppID(32byte) +sourceIntanceId(16byte)
 *                      +MsgType(8byte) +Flag(8byte) +BizType(8byte)
+Priority(8byte) +SensitiveLevel(8byte)
 *                      +UserData1(256byte) +UserData2(256byte) +data
 *                      errcode= "0000"-success，errcode !="0000"-fail；
 *            fail:"NULL" or "NULL，errmsg"；
 * note：parameters sAppId and iInstanceId are needed；
 */
public native static byte[] Mr3Receive3(String sAppId, int iInstanceId,String
sSourceUserID,String sSourceAppID, String sDestUserID, String sDestAppID,
                                         String sPkgID, String sCorrPkgID, String
sUserData1, String sUserData2, int iMillSecTimeo);

/*
 * return: PkgId, empty:fail；not empty:success
 * note：parameters sSourceAppID and iSrcInstanceId are needed；
*/
public native static String Mr3SendTopicMsg( byte[] psPkg, String sSourceUserID,String
sSourceAppID, int iSrcInstanceId, String sTopicName,
                                   String sPkgID, String sCorrPkgID, String
sUserData1, String sUserData2, byte cFlag, byte cPriority, byte cSensitiveLevel);

/*
 * return：success:  pkID(64byte) +CorrpkID(64byte) + topicName(17byte)
+sourceUserID(32byte) +sourceAppID(32byte) +sourceInstanceID(16byte)
 *                    +flag(8byte) +Priority(8byte) +SensitiveLevel(8byte)+
UserData1(256byte) + UserData2(256byte) +data
 *           fail: "NULL" or "NULL，errmsg";
 * note：parameters sAppId and iInstanceId are needed；
*/
public native static byte[] Mr3ReceiveTopicMsg(String sAppId, int iInstanceId, int
iBufferLenIn, String sSourceUserID,String sSourceAppID, String sTopicName);
```

```
/*
 * return: string success：taskid；fail：empty
 * note:parameters sSourceAppID and iSrcInstanceID are needed；
*/
public native static String Mr3SendFile(String sFileFullPath, String sReserveInfo,String
sSubDir, String sSourceUserID,String sSourceAppID,
                                                  int iSrcInstanceID, String sDestUserID, String
sDestAppID, int iDestInstanceID, String sUserData1, String sUserData2, int iBizType);

/*
 * return: success：taskID(128byte) +srcFileName(496byte) +destFileName(496byte)
+sourceUserID(32byte) +sourceAppID(32byte) +sourceIntanceId(16byte)
+bizType(8byte) + UserData1(256byte) + UserData2(256byte) + ReserveInfo
 *            fail："NULL" or "NULL,errmsg"
 * note：parameters sAppId and iInstanceId are needed；
*/
public native static byte[] Mr3ReceiveFile(String sAppId, int iInstanceId);

/*
 * return:   success：FileStatus(8byte) +Percent(8byte) +FileThoughOutKbps(8byte)
 *             fail："NULL" or "NULL,errmsg"
 * note：parameters sAppId and iInstanceId are needed；
*/
public native static byte[] Mr3GetSendFileStatus(String sAppId, int iInstanceId, String
sTaskId);

/*
 * return: 0:success；other-fail
 * note：parameters sAppId and iInstanceId are needed
*/
public native static int Mr3CancelSendFile(String sAppId, int iInstanceId, String
sTaskId);

/*
 * return success：AppID(32byte) +LinkID(32byte) +InstanceID(16byte) +IP(16byte)
 *           +Port(8byte) +MrName(32byte) +LoginTime(256byte)
+QueueLength(64byte) +TopicQueueLength(64byte)
 *           +FileQueueLength(64byte) +SendPkgCountTotal(64byte)
+RecvPkgCountTotal(64byte)
 *           fail："NULL" or "NULL,errmsg"
 * note：1、multiple status are separated by “|”；
 *          2、sQueryAppId：empty for all app；
 *          3、parameters sAppId and iInstanceId are needed；
*/
```

```
public native static  byte[] Mr3GetAppStatus(String sAppId, int iInstanceId, String
sQueryAppId);

}
```

After writing the mrapi.java file, compile the file using the javac command to generate the mrapi.class file in the fdep directory.

7.2.2 Sample code

After generating the mrapi.class file, you can use the mrapi class method to use the various functions described in section 4.3 of this manual. The sample code is as follows (the sample code is provided in the Demo.java file in the rar package)：

```
/**
 * Created by sscc on 2020/1/13.
 */


import com.sscc.fdep.*;

import java.io.UnsupportedEncodingException;

public class MsgDemo extends Thread
{
    //Main thread: initialization
    public static void main( String agrs[])
    {
        CfgParam oCfgParam = new CfgParam();
        run(oCfgParam);
        mrapi.Mr3Destroy(oCfgParam.m_ssAppID,
oCfgParam.m_usSourceInstanceID);
    };

    public static void run(CfgParam oCfgParam)
    {
         //Call Mr3Init to initialize
        int iErr = mrapi.Mr3Init(oCfgParam.m_ssAppID,
oCfgParam.m_usSourceInstanceID, oCfgParam.m_ssAppPasswd,
oCfgParam.m_ssMrIP1, oCfgParam.m_usMrPort1, oCfgParam.m_ssMrIP2,
oCfgParam.m_usMrPort2);
        if (iErr == 0)
        {
            System.out.println("ERROR: Mr3Init failed, exit.");
            return;
        }
        else
        {
            System.out.println("Mr3Init OK.");
        }

        byte[] Version = mrapi.Mr3GetVersion();
        String sVersion = null;
        try {
            sVersion = new String(Version, "GBK");
        } catch (UnsupportedEncodingException e) {
            e.printStackTrace();
        }
        System.out.println("Mr3GetVersion[" + sVersion + "]");
        //Start send thread
        Runnable RunSendThrd = new SendThrd(oCfgParam);
        Thread SndTd = new Thread( RunSendThrd );
        SndTd.start();
        //Start recv thread
        Runnable RunRecvThrd = new RecvThrd(oCfgParam);
        Thread RcvTd = new Thread( RunRecvThrd );
        RcvTd.start();

        while(true)
        {
            //Call Mr3IsLinkOK every 1s to determine whether the AppID is connected
to mr normally: if it is not normal, it will alarm. After the network connection is normal,
mrapi will automatically reconnect to mr
            if (mrapi.Mr3IsLinkOK(oCfgParam.m_ssAppID,
oCfgParam.m_usSourceInstanceID) == 0)
            {
                System.out.println("WARN: Link not ok.");
            }

            try
            {
                sleep(1000);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}

//Configuration parameter
class CfgParam
{
    public String m_ssAppID;
    public String m_ssAppPasswd;
    public String m_ssMrIP1;
    public short   m_usMrPort1;
    public String m_ssMrIP2;
    public short   m_usMrPort2;
    public String m_ssSourceUserID;
    public short   m_usSourceInstanceID;
    public String m_ssDestUserID;
    public String m_ssDestAppID;
    public short   m_usDestInstanceID;
    public String m_ssTopicName;
    public String m_ssUserData1;
    public String m_ssUserData2;
    public byte m_ucFlag;
    public byte m_ucBizType;

    public CfgParam()
    {
        m_ssAppID = "app1";
        m_ssAppPasswd = "1";
        m_ssMrIP1 = "10.10.218.64";
        m_usMrPort1 = 5852;
        m_ssMrIP2 = "10.10.218.64";
        m_usMrPort2 = 5852;
        m_ssSourceUserID = "FTCSTEST1021";
        m_usSourceInstanceID = 100;
        m_ssDestUserID = "FTCSTEST1020";
        m_ssDestAppID = "app1";
        m_usDestInstanceID = 0;
        m_ssTopicName = "1021sscc";
        m_ssUserData1 = "UserData1";
        m_ssUserData2 = "UserData2";
        m_ucFlag = 0;
        m_ucBizType = 0;
    }
}
```

```
//Send thread
class SendThrd extends Thread
{
    private CfgParam m_oCfgParam;
    public static int m_iCount = 0;

    public static String bytesToString(byte[] b)
    {
        StringBuffer result = new StringBuffer("");
        int length = b.length;
        for (int i = 0; i < length; i++)
        {
            result.append((char)(b[i] & 0xff));
        }
        return result.toString();
    }

    public SendThrd(CfgParam oCfgParam)
    {
        this.m_oCfgParam = oCfgParam;
    }

    public void run()
    {
        while(true)
        {
            int iPeerStat = mrapi.Mr3GetPeerUserStat(m_oCfgParam.m_ssAppID,
m_oCfgParam.m_usSourceInstanceID, m_oCfgParam.m_ssDestUserID);
            System.out.println("PeerUser[" +m_oCfgParam.m_ssDestUserID +"]
Stat["+ iPeerStat + "]");

            byte[] szCreatePkgId =
mrapi.Mr3CreatePkgID(m_oCfgParam.m_ssAppID,m_oCfgParam.m_usSourceInstance
ID);
            String ssCreatePkgId = bytesToString(szCreatePkgId);
            System.out.println("PkgId ["+ssCreatePkgId + "]");
            //Data sent
            byte[] senddata = new byte[] { ('H'), ('E'), ('L'), ('L'), ('O') };
           //Call Mr3Send to send to the receiver, the return value is PkgID
            String ssPkgID = mrapi.Mr3Send(senddata,
m_oCfgParam.m_ssSourceUserID,
m_oCfgParam.m_ssAppID,m_oCfgParam.m_usSourceInstanceID,
m_oCfgParam.m_ssDestUserID, m_oCfgParam.m_ssDestAppID,
                    "", "", m_oCfgParam.m_ssUserData1,
m_oCfgParam.m_ssUserData2, "", m_oCfgParam.m_ucFlag,
m_oCfgParam.m_ucBizType, (byte)0,(byte)0,2000);
            System.out.println(ssPkgID);
            if (ssPkgID==null||ssPkgID.length()<=0)
            {
                System.out.println("ERROR: send failed: PkgID[],
src["+m_oCfgParam.m_ssSourceUserID+"."+m_oCfgParam.m_ssAppID+"],
dest["+m_oCfgParam.m_ssDestUserID+"."+m_oCfgParam.m_ssDestAppID+"]");
            }
            else
            {
                System.out.println("send ok: PkgID["+ssPkgID+"],
src["+m_oCfgParam.m_ssSourceUserID+"."+m_oCfgParam.m_ssAppID+"],
dest["+m_oCfgParam.m_ssDestUserID+"."+m_oCfgParam.m_ssDestAppID+"]");
            }
            //Send every 1s
            try
            {
                sleep(1000);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}

//Receiving thread
class RecvThrd extends Thread
{
    public static String bytesToString(byte[] b)
    {
        StringBuffer result = new StringBuffer("");
        int length = b.length;
        for (int i = 0; i < length; i++)
        {
            result.append((char)(b[i] & 0xff));
        }
        return result.toString();
    }

    private CfgParam m_oCfgParam;
    public RecvThrd(CfgParam oCfgParam)
    {
        this.m_oCfgParam = oCfgParam;
    }

    public void run()
    {
        //int iRegRet =
mrapi.Mr3RegRecvCondition(m_oCfgParam.m_ssAppID,m_oCfgParam.m_usSourceIns
tanceID, "", "", "", "", "", "", "", "") ;
        //System.out.println("Mr3RegRecvCondition=" + iRegRet);

        while(true)
        {
           //Call Mr3Receive1 to receive the message packet sent by the other party
            byte[] recvdata =
mrapi.Mr3Receive1(m_oCfgParam.m_ssAppID,m_oCfgParam.m_usSourceInstanceID,
"", "","", "", "", "", "","", 2000);
            String result = bytesToString(recvdata);
            if(!result.equals("NULL\0") && !result.contains("NULL"))
            {
                //Received a packet
                //Intercept the required field data: recvdata = pkID(64byte)
+CorrpkID(64byte) +sourceUserID(32byte) +sourceAppID(32byte)
+sourceIntanceId(16byte)
                //+MsgType(8byte) +Flag(8byte) +BizType(8byte) +Priority(8byte)
+SensitiveLevel(8byte)
                //+UserData1(256byte) +UserData2(256byte) +data
                int iStartIdx = 0;
                int iEndIdx = iStartIdx+64;
                String ssPkgID = result.substring(iStartIdx,iEndIdx).trim(); //pkID

                iStartIdx = iEndIdx;
                iEndIdx = iStartIdx+64;
                String ssCorrPkgID = result.substring(iStartIdx,iEndIdx).trim();
//CorrpkID

                iStartIdx = iEndIdx;
                iEndIdx = iStartIdx+32;
                String ssSourceUserID = result.substring(iStartIdx,iEndIdx).trim();
//sourceUserID

                iStartIdx = iEndIdx;
```

```
                    iEndIdx = iStartIdx+32;
                    String ssSourceAppID = result.substring(iStartIdx,iEndIdx).trim();
//sourceAppID

                    iStartIdx = iEndIdx;
                    iEndIdx = iStartIdx+16;
                    String ssIntanceId = result.substring(iStartIdx,iEndIdx).trim();
//sourceIntanceId

                    iStartIdx = iEndIdx;
                    iEndIdx = iStartIdx+8;
                    String ssMsgType = result.substring(iStartIdx,iEndIdx).trim();
//MsgType

                    iStartIdx = iEndIdx;
                    iEndIdx = iStartIdx+8;
                    String ssFlag = result.substring(iStartIdx,iEndIdx).trim(); //Flag

                    iStartIdx = iEndIdx;
                    iEndIdx = iStartIdx+8;
                    String ssBizType = result.substring(iStartIdx,iEndIdx).trim();
//BizType

                    iStartIdx = iEndIdx;
                    iEndIdx = iStartIdx+8;
                    String ssPriority = result.substring(iStartIdx,iEndIdx).trim(); //Priority

                    iStartIdx = iEndIdx;
                    iEndIdx = iStartIdx+8;
                    String ssSensitiveLevel = result.substring(iStartIdx,iEndIdx).trim();
//SensitiveLevel

                    iStartIdx = iEndIdx;
                    iEndIdx = iStartIdx+256;
                    String ssUserData1 = result.substring(iStartIdx,iEndIdx).trim();
//UserData1

                    iStartIdx = iEndIdx;
                    iEndIdx = iStartIdx+256;
                    String ssUserData2 = result.substring(iStartIdx,iEndIdx).trim();
//UserData2
                    //Received data
                    String ssData = result.substring(iEndIdx);
                    System.out.println("recv ok: PkgContent[" + ssData + "]");
                    System.out.println("pkgID[" + ssPkgID + "], src[" + ssSourceUserID +
"." + ssSourceAppID + "."+ssIntanceId+ "],   pkgLen[" + ssData.length() + "],
CorrPkgID[" + ssCorrPkgID + "], UserData1[" + ssUserData1 + "], UserData2[" +
ssUserData2 + "]");
                    System.out.println("MsgType[" + ssMsgType + "], Flag[" + ssFlag + "],
BizType[" + ssBizType + "], Priority[" + ssPriority + "], SensitiveLevel[" +
ssSensitiveLevel + "]");
                   //Determine whether it is a request or response
                    if (ssCorrPkgID==null||ssCorrPkgID.length()<=0)
                    {
                         //General situation: If CorrPkgID is empty, it means the message
packet is a request packet; if it is not empty, it means that the message packet is a
response packet.
// Process the request packet: The following processing is recommended to throw the
message packet to the business processing thread for processing.
// ...
// Construct the response packet data and call Mr3Send function to send the response
packet.
                         byte[] PesponsePkg = new byte[] { ('R'), ('e'), ('s'), ('p'), ('o'), ('n'),
('s'), ('e'), (' '), ('p'), ('k'), ('g') };
                        //Return response packet: Assign the value of the response packet
attribute CorrPkgID to the value of the corresponding request packet attribute PkgID.
mrapi.Mr3Send(PesponsePkg, m_oCfgParam.m_ssSourceUserID,
m_oCfgParam.m_ssAppID,m_oCfgParam.m_usSourceInstanceID, ssSourceUserID,
ssSourceAppID,
                                    "", ssPkgID, ssUserData1, ssUserData2, "",
m_oCfgParam.m_ucFlag, m_oCfgParam.m_ucBizType, (byte)0,(byte)0,2000);
                    }
                    else
                    {
//General situation: If m_szCorrPkgID is empty, it means that the message packet is a
request packet; if it is not empty, it means that the message packet is a response
packet.
// Process the response packet: The following processing is recommended to throw the
message packet to the business processing thread for processing.
// ...
                     }
// Continue to receive the next message packet
                    continue;
               }

//No packet, sleep for a while
               try
```

```
            {
                sleep(20);
            }
            catch (InterruptedException e)
            {
                e.printStackTrace();
            }
        }
    }
}
```

Compile and execute the MsgDemo .java file, you can view the program execution effect.

8 Backup Solution

The API of the financial data exchange platform provided by SSCC is called FDEAPI. FDEAPI provides one main IP address and two IP addresses and ports. When the API fails to connect to one BSMR software, it will automatically connect to the other BSMR software. In this way, when the BSMR software fails, the user's application system can connect to the BSMR for normal work without any changes.

In addition, the main function of the API of the financial data exchange platform message transmission system is to send and receive messages. For sending messages, the API can be sent from any machine; for receiving messages, the financial data exchange platform API provides a way to actively get to BSMR according to the conditions, and can only get the message packets that it needs. Therefore, the API of the financial data exchange platform can support application load balancing and multi-machine parallel processing. However, whether the user's application system is determined to support load balancing and multi-machine parallel processing is related to the program provided by the software developer. The user should consult the software developer calling FDEAPI for relevant technical details.

9 Precautions

1. In the Windows environment, the ini directory should be placed in the same directory as mrapi.dll and in the non-Windows platform environment, placed according to the hierarchical structure after unpacking. If the configuration file mrapi.ini is not in the same level directory of the mrapi dynamic link library, it is required to set the environment variable

"MRAPI_LOGCONF_PATH" to the absolute path of the configuration file mrapi.ini. If the configuration file does not exist, the startup will fail.

2. In the development process, only one of three methods MrXXXX or Mr2XXXX or Mr3XXXX can be used, the three cannot be used together in the development process.

3. In a non-Windows platform environment, configure environment variables to ensure that the project directory is added to LD_LIBRARY_PATH.

10 FAQ

10.1 Using Long Connections

In the process of using FDEAPI, the correct usage is that users should use long connections. That is, the MrInit function (connection establishment) is generally called only in the initialization part of the program, and then normal packet sending and receiving operations are performed, and MrDestroy is called when the program exits. Instead of calling MrInit once every time a request comes in, release after processing.

10.2 About the use of CorrPkgID

If the user calls the MrSend function to send a business response packet, for the m_szCorrPkgID field in the STUMsgProperty parameter, it is agreed to fill in the same value as the m_szPkgID field in the corresponding request packet, so that the requester receives the response packet according to this field. Find the corresponding request packet. For service request packets or other types of packets, this field is blank (ie, '\ 0').

When receiving, if the value of the m_szCorrPkgID field in STUMsgProperty is empty, it means a request packet; otherwise it means a response packet.

10.3 If it is synchronous processing, when a request is issued, it needs to wait for reception in a loop

If the user uses the synchronous receiving method for the packet sent to the FDEP message transmission system , that is, after sending it to the FDEP message transmission system through MrSend, then wait for the response method to be received. For this processing method, you need to pay attention to it during development. If you receive it immediately after sending it, you can be sure that you cannot receive the response packet for the first time, because the response packet cannot be as fast as running on the local machine. You can come back, you have to wait for a certain time before you can receive it. The pseudo code is as follows:

```
MrSend(); //send
time_t  ltBeginSecond = time(NULL);  //Get the current time in seconds
while(1)
{
        int iRet = MrReceive();
        if(iRet==0) break;  //recv，OK
        //For each cycle, the current time is subtracted from the start reception time. If it
is greater than 20 seconds, it indicates a timeout.
if(time(NULL)- ltBeginSecond>20) break;
}
```

10.4 App1 and app2 conventions

Generally, the appID of the user receiving the request is app1. Use any appID to send a request packet to the other's app1. After receiving the request, the other's app1 processes it. After the processing is complete, the response is sent back to the requester according to the source UserID and appID of the requester.

10.5 The API is thread-safe, but not process-safe

The API is thread-safe, but not process-safe. Once a handle is created by calling MrInit, the handle will not change until MrDestroy is called. The same handle can be used between multiple threads in the same process, but cannot be used between different processes. For example, after a fork under Unix, the handle created in the main process can no longer be used.

Shenzhen Securities Communications Co., Ltd.

April 2020