2017/10

Ver. 1.07

MXNET2 REMOTE API MANUAL

Contents

1. Distribution Files

+x64 (64bit Server & Client API)

+clientlib

Mxnet2_api.dll	API DLL
Mxnet2_api.lib	mxnet2_api.dll import library for VC
Mxnet2st_mt.lib	API VC static library(/MT) for VC
Mxnet2st_md.lib	API VC static library(/MD) for VC
Mxnetapp.ini	config file for client API

+server

Mxnet2.exe	mxnet2 server
Mxnet2_ui.exe	UI program for server
Mxnet2_detect.exe	server failure protection program
Mxnet2.ini	config file for mxnet2 server
Alerm.wav	sound file for alert

+x86 (32bit Server & Client API)

the same files as under x64

+Samples

+C Sample

testDLL.sln	VS2013 Solution File

+C# Sample

testcsharp.sln	VS2013 Solution File

Userpass.exe

Userpass.ini

2. Server Installation

The distribution ZIP file contains both 32bit and 64bit versions of the remote API servers and clients. On 32bit Windows, use 32bit version. On 64bit Windows, you can run either versions.

To install mxnet2 server, copy the following 4 files in “server” folder to the computer that functions as mxnet2 server. Mxnet2 server may reside on the same computer as the client programs. All 4 files must be copied to the same folder.

1. Mxnet2.exe (mxnet2)
2. Mxnet2_ui.exe (GUI for mxnet2 service)
3. MxNet2_detect.exe (failure protection service)
4. Mxnet2.ini (config. file for mxnet1)

3. MxNet2 Server Configuration File

Mxnet2 is a network server that accepts requests from remote API clients. When run, MxNet2 server reads the configuration file named mxnet2.ini in the same folder. The default port number is 12300. The default value at the client side is also 12300. In a test environment, you may run mxnet2 out of box. In a production environment, we suggest you modify the port value to accommodate your runtime environment.

3.1 mxnet2.ini

```
[Option]
#Language setting
Lang=eng

# the port mxnet1 waits on
Port=12300
```

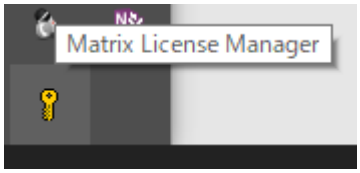
```
# LogIn_MatrixNet Timeout in second
LoginTimeOut=300

#Init_MatrixAPI Timeout in second
SessionTimeOut=3000

#License type
# AppSlot license is either per IP or per session
SessionBaseLicense=1

#LogFile folder
LogFileFolder=
LogLevel=0
```

4. Running MxNet2 Server

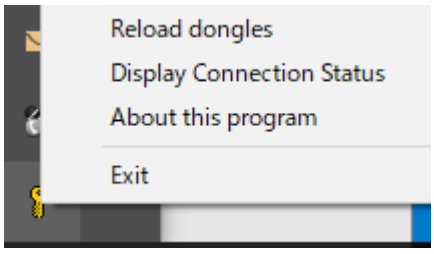
Mxnet2.exe may be run either as an application program or as a Windows service.

4.1 Running as an application program

Just double-click Mxnet2.exe and it will display a yellow key icon in the system tray.



a. To terminate the program, right-click the icon and select “Exit”



b. “Display Connection Status” will display a window showing the details on the

currently connected clients and login status. In this window, you can manually disconnect active sessions and active logins to AppSlot.

4.2 Running as a Windows service

Firewall Setting

Before running Mxnet2 as a Windows service, be sure that the port Mxnet2 waits on is not blocked by the firewall. Even if the port is blocked, Mxnet2 service will run normally; no client can connect to it.

You must register Mxnet2 as a Windows service to run it as a service. For the service registration, you must be elevated to an administrative user. Open a command prompt as Administrator, change the current directory to where mxnet2.exe is located and type the following command.

- Mxnet2 install [enter]

This will register Mxnet2 as a Windows service and run it as a Windows service.

Important：Registering a program as a Windows service will only save the program path in the registry. Do not move or delete the program file. If you do, the registered service will not run. Do not place Mxnet2 on removable disks; they are usually not ready for use at the time Windows starts services.

The running instance of mxnet2 service will display mxnet2 yellow icon in the system tray just as mxnet2 is run as an application program. This icon is displayed, not by the Windows service, but by another program, mxnet2_ui.exe. This UI program communicates with mxnet2 service. By right-clicking the icon, you can exit the program. This does not stop or terminate Mxnet2 service; it just terminates the UI program. While the service is running, you can run mxnet2_ui.exe manually at any time.

MxNet2 as a Windows service can be de-registered by the following command. Open a command prompt as Administrator and run the command.

- Mxnet1 remove [enter]

5. MxNet2 Server Fault Tolerance

To deliver high availability, mxnet2 comes with another service program that ensures that mxnet2 be always running. This auxiliary service continually queries Mxnet2 for response. When no response is returned, it automatically restarts mxnet2 service.

The service program file is named Mxnet1_detect.exe. For the program to restart mxnet2 server, it must be in the same folder as MxNet2.exe.

When Mxnet2_detect re-starts MxNet2, no response to a query might be returned due to critical failures. This signals mxnet2 failure. If the auxiliary service cannot restart Mxnet2 successfully, it will play the sound file named alerm.wav in the same folder for audible alert till you stop the service or manually recover MxNet2 server to a running state.

To register Mxnet1_detect as a Windows service: >mxnet1_detect -i

To remove Mxnet1_detect as a Windows Service: >mxnet1_detect -r

You do not have to start/stop both MxNet2 and the auxiliary service manually. Stopping mxnet2_detect service will automatically stop mxnet2 server. Starting xnet2_detect service will likewise start mxnet2 server.

6. How MxNet2 Server Works

You can call Remote API just the same way as Matrix API.

But unlike Matrix API client that directly interacts with the dongle, Remote API clients communicate with the remote Mxnet2 server over a communication channel. This can complicate the use of remote API.

6.1 rInit_MatrixAPI & rRelease_MatrixAPI

In Remote API, a call to rInit_MatrixAPI will start a new session with Mxnet2 server. rRelease_MatrixAPI will release/close the session. If you do not start a session by a call to rInit_MatrixAPI, all the other APIs will fail. The server only accepts requests from a

client with an open session to the server.

By calling rInit_MatrixAPI, the client establishes a session (one like HTTP session) with the server. It saves the client information and returns a session identifier to the client. Any subsequent remote API calls by the client will internally carry this identifier. rRelease_MatrixAPI call will de-allocate the client session information on the server and invalidates the session identifier. If a client does not call rRelease_MatrixAPI before terminating, the client's session information will be held alive on the server till the session timeout handler deletes it.

6.2 Session

A session is started by rInit_MatrixAPI. Every time a client calls rInit_MatrixAPI, a new session is started.

1. The server collects the information on the client.
2. The server and a client agree on the session password
3. The server assigns a session ID to the client
4. A session is released by rRelease_MatrixAPI

6.3 Session Timeout

After a session is started, each API call will update the session's last access time on the server. A session will time out if the difference between the last access time and the current time is over the specified session timeout value. If the timeout value is 180 sec (3 min) and a client does not call any remote API for over 180 sec, then, the session will be closed. The next API call to the server by the client will fail unless the client calls rInit_MatrixAPI and starts a new session.

When the session timeout value is too small, the server may release a session while the client is still alive. When the session timeout value is too big, zombie sessions will live on long. Zombie sessions do no harm to the server. But they makes extremely difficult (nearly impossible) to tell which sessions are actually alive or dead.

The default value for the session timeout is 3600sec(1 hour).

Zombie sessions can be removed manually in "Display Connection Status" window, by

selecting a session and press “Logout” button.

6.4 Remote API Session vs. Communication Session

The session started by rInit_MatrixAPI is different from the communication session. Mxnet2 session is just like http session. Each API will open a communication channel, send a command to the server, receive a response and shut down the communication channel. Communication sessions are not kept open between API calls.

Remote API session refers to the state in which the server can identify a client between API calls by holding the common data with the client. When the server one-sidedly releases the session data of a live client, the session is erased; the server no longer can identify the client. In such a case, the client must start a new session by calling rInit_MatrixAPI or any other API call will fail.

Before a client terminates, be sure to close an open session; it will immediately release the client session data from the server and avoid zombie session being kept alive on the server.

6.5 rInit_MatrixAPI & rRelease_MatrixAPI in a Multi-threaded Program

A session is created for a process; it is not initiated for every thread that calls rInit_MatrixAPI. Once a session is created for a process, all threads in the process will share one(1) open session.
When multiple threads in a process call rInit_MatrixAPI, the first rInit_MatrixAPI call will be sent to the server and a new session is created. While the session is alive, any subsequent calls to rInit_MatrixAPI by any threads in the process will not start a new session. Such rInit_MatrixAPI calls will not be even sent to the server; only the internal counter in the client API library is incremented. The internal counter keeps the call count of rInit_MatrixAPI.
Conversely, a call to rRelease_MatrixAPI will only decrement the internal counter by one when the internal counter is greater than 1. A call to rRelease_MatrixAPI, when the counter is equal to 1, will be sent to the server and releases the client session.

Resetting the internal counter

In some cases, you may have to reset the internal counter. Suppose that, after 2 threads have called Init_MatrixAPI which have set the internal counter value to 2, the client's session data on the server is inadvertently released. The client no longer has a session with a server. It must start a new session by calling rInit_MatrixAPI. But with the internal counter set to 2, no new session can be created; it will only increment the counter to 3. The internal counter must be 0 to start a new session.

One way to reset the counter would be to keep calling rRelease_MatrixAPI till the return value is 0. Alternatively, you can all the counter reset API, Reset_MatrixAPI(). This will unconditionally reset the counter to 0. After the call to the reset API, rInit_MatrixAPI will start a new session

```
void __stdcall rReset_MatrixAPI(void);

[DllImport("mxnet1_api.DLL", EntryPoint = "rReset_MatrixAPI", CallingConvention =
CallingConvention.StdCall)]
public static extern void rReset_MatrixAPI();
```

6.6 License Count Management API rLogIn_MatrixNet & rLogOut_MatrixNet

Remote API has rLogIn_MatrixNet AP. It works the same way as Matrix LogIn_MatrixNet. Matrix dongle's memory fields(AppSlot) hold license counts. A call to LogIn_MatrixNet will decrement the specified AppSlot's license count by one. The next call from the same process to LogIn_MatrixNet will not decrement the license count; this time, it will only refresh the login to the same AppSlot. Once logged in, the client must keep calling LogIn_MatrixNet periodically to hold on the license acquired by the first call to LogIn_MatrixNet. If it fails to refresh the login within a specific timeout interval, the login timeout handler will remove the client login and increment the AppSlot license count(the client is logged out)

A call to rLogOut_MatrixNet will increment the specified AppSlot's license count.

The login timeout handler prevents license count exhaustion by dead sessions. If zombie sessions continue to hold the licenses, a live session will not be able to acquire a new license. The login timeout handler periodically scans the last login refresh time of each session and log off the sessions that have not refreshed the login by a call to LogIn_MatrixNet within a timeout interval.

After the first call rLogIn_MatrixNet to acquire a license, a session must keep calling rLogIn_MatrixNet periodically to avoid losing the license. The default value of the timeout is 300sec(5 min). You need to call rLogIn_MatrixNet at least once in 300 sec.

The above description shows that you do not call rLogIn_MatrixNet and rLogOut_MatrixNet in pair, unlike open and close. Multiple calls to rLogIn_MatrixNet at a specified AppSlot will only decrement the license count by one while a call to rLogOut_MatrixNet will always increment the license count by one.

Example:
A thread A in a multi-threaded program calls rLogIn_MatrixNet on AppSlot10. The process will acquire a license from the AppSlot. Another call to rLogIn_MatrixNet by another thread B will only refresh the login. A little later, the thread B calls rLogOut_MatrixNet and the process will release the license. Another call to rLogOut_MatrixNet by the thread A will result in error(-11), because the process no longer has a license at that point.

6.7 Naming AppSlots

“Display Connection Status” window shows the list of which clients currently have sessions and which AppSlots each client logs in to.

By default, mxnet2 shows Dongle and AppSlot as “DongleX” and “AppSlotX” where x is a number like

Client3
Dongle1 AppSlot5
Dongle1 AppSlot6

In this example, Client3 has a session and it has acquired a license each from AppSlot5 and AppSlot6 of Dongle1 – the first dongle detected by mxnet2.

You can assign arbitrary strings to dongles and AppSlots to help identify them. After you name them, mxnet2 will display the assigned string instead of DongleX/AppSlotX.

Client3
MyDongle FooApplication
MyDongle BarApplication

To name a dongle and its AppSlot, create a section named [dongleX] where X is a number starting from 1. Under the section, create SerNr, Name and Slot Number keys.

```
[dongle1]
Sernr=1234567890
Name=MyDongle
5=Foo Application                    #AppSlot5 is named “Foo Application”
6=Bar Application                    #AppSlot6 “Bar Application”

[dongle2]
Sernr=1234567891
Name=MyDongle1
10= App10
11= App11
```

“Sernr” and “Name” keys are optional. mxnet2 searches [DongleX] sections and sees if there is a section with “Sernr” key whose values matches the current dongle. If it finds one, mxnet2 uses the section for string assignment. If it does not, mxnet2 uses the section with the value of “DongleX” where x is DngNr of the current dongle. If it does not find any section, then the dongle name will be “DongleX” (where X is DngNr) and “AppSlotX” (where x is the slot number).

[dongleX] is the section name wherer x increments sequentially. Do not skip a number. Mxnet2 tries to read DongleX section sequentially and stops reading when the next number is not found.

```
[dongle1]
[dongle2]
[dongle4]
```

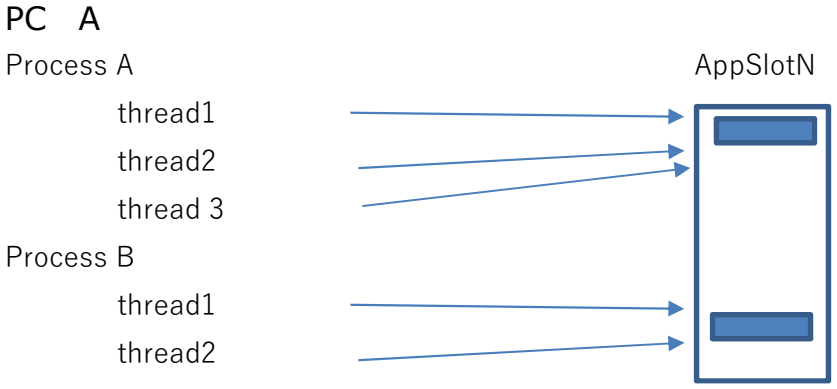
Mxnet2 will read the first two(2) sections and stop reading. It will not read [dongle4].

6.7 License Type : session or IP

MxNet2 will use either the session identifier or IP address to identify a client for the login purpose. By default, it uses the session identifier for client's identity. MxNet2.ini has SessionBaseLicense=1 as the default.

a. Session based License SessionBaseLicense=1

A process (session) will acquire a license from a specified AppSlot. When multiple processes are running on a computer, each process will decrement the license count of a AppSlot.



- Thread 1 in the process A on PC calls rLogIn_MatrixNet. It will decrement the AppSlot counter. Client API library's internal counter is also incremented.
- Thread 2 in the process A on PC calls rLogIn_MatrixNet. No request is sent to the server. Client API library's internal counter is incremented to 2.
- Thread 3 in the process A on PC calls rLogIn_MatrixNet. No request is sent to the server. Client API library's internal counter is incremented to 3.

- Thread 2 in the process A on PC calls rLogOut_MatrixNet. No request is sent to the server. Client API library's internal counter is decremented to 2.
- Thread 3 in the process A on PC calls rLogOut_MatrixNet. No request is sent to the server. Client API library's internal counter is decremented to 1.

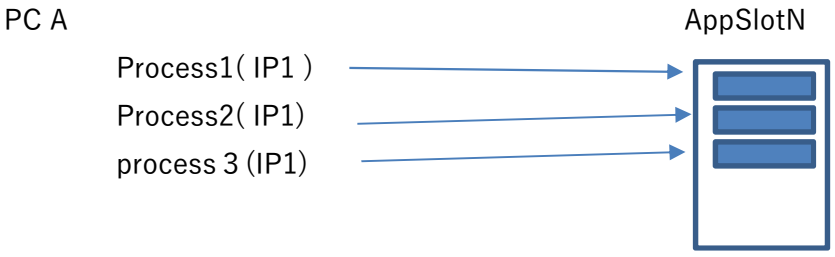
- Thread 1 in the process A on PC calls rLogOut_MatrixNet. A request is sent to the server and the session is closed. The Internal counter is also decremented to 0.

- Threads in Process B behaves the same way

b. IP based License SessionBaseLicense=0

Mxnet2 will identify clients by IP address. When multiple processes are running on a computer and they call rLogIn_MatrixNet, the first call by any processes on the computer will decrement AppSlot license and the subsequent calls by any processes will refresh the login. The requests from the processes on the same computer have the same IP address and mxnet2 treats them as if they were from the same client.

rLogOut_MatrixNet works differently from Session based license scheme. When multiple processes on the same computer calls rLogIn_MatrixNet, mxnet2 server keeps the record of each client. AppSlot count is managed not by the session identifier but by IP address. MxNet2 scans the record of all client sessions and if there is no session with the same IP, AppSlot license is decremented by one. If a session with the same IP exists, MxNet2 refreshes the login time. When a client calls rLogOut_MatrixNet, it will scan the existing client session record to see if there is a session with the same IP; if there is, it will not not increment AppSlot license. Mxnet2 increments AppSlot license when a request comes from a client whose IP is not found in any other login records.



- Process 1 on PC calls rLogIn_MatrixNet. It will decrement AppSlot license
- Process 2 on PC calls rLogIn_MatrixNet. The call will only refresh IP1's login. Mxnet2 will save the record of process2's login
- Process3 on PC calls rLogIn_MatrixNet. The call will only refresh the IP1's login. Mxnet2 will save the record of process3's login

- Process2 on PC calls rLogOut_MatrixNet. Since there are other login record with IP1, this will only remove Process2's login record

- Process1 on PC calls rLogOut_MatrixNet. There are still login records with IP1. This will only delete Process1's login record.
- Process3 on PC calls rLogOut_MatrixNet. Process3's record is the only login record with IP1. This will remove Process3's record and AppSlot license count is incremented
-

Just with Session Based License, rLogIn_MatrixNet / rLogOut_MatrixNet calls by each process may or may not be sent to the server; Client API's internal counter keeps the call count.

6.8 Clock synchronization between the server and the clients

The requests from the clients to the server are time-stamped. The server will try to verify that the client request is issued within a specified time interval. If the request is more than 60 seconds old, the server will reject the request.

You can specify the command interval time by **CmdIssueTimeInterval** key in the server configuration. Give a time value in second to the key

If the server clock and the client clock cannot be synchronized, you can disable the request time-stamp verification. Give 0 to CheckCmdIssueTime in the server configuration file.

Mxnet2.ini
[option]
CheckCmdIssueTime=0

6.9 Server Log File

Mxnet2 can create log files in a specified folder. In the server configuration file, specify the folder name using LogFileFolder key.

[Option]

LogFileFolder=c:¥folder
LogLevel=0

Mxnet2 will create log files whose name will be in the following format.

mxnetYYYY-MM-DD.log

where YYYY-MM-DD is the date when the logs are written. Mxnet2 creates different log files on each different date.

By default, mxnet2 will log the error records. By specifying LogLevel in the configuration, more details logs may be recorded

[option]
LogLevel=1 information log
LogLevel=2 debug log

7. MxNet2 Error

Error No	Description
-100	Connect Error Cannot connect to MxNet2 server
-102	No Matrix dongle connected
-104	Request time-stamp error
-105	Session password error
-106	Packet length error
-107	Packet header error
-108	Unknow Command
-115	Packet Number error
-116	Client library does not have a session with the server
-122	Packet version error

This is not an exhaustive list of errors.

Error -100 is the most common error. It is returned when the client cannot connect to the server. This is returned for various reasons; the server may not be running, the port on the server computer is closed, the ports specified by the client and by the server is different, the server IP address specified by the client is not correct, etc.

8. Client API

A program that integrates Remote API must use DLL or static client libraries.

7.1 RemoteAPI DLL(mxnet1_api.dll)
If your program is written in a language that require the use of DLL, you can use DLL as provided or create a wrapper DLL using the static libraries. The API in the supplied DLL conforms to __pascal calling convention; some languages may require DLL functions to follow _cdecl convention.

.NET program can import the supplied DLL using InteropServices. Please refer to C# sample.

Configuration file for API DLL
When DLL is loaded, it will read the configuration file, mxnetapi.ini, in the same folder as the DLL. It uses the server IP address and the port specified in the configuration file.

```
[Options]
# MxNet2 server IP address
ip=127.0.0.1

# MxNet2 Server Port
port=12300

# Connect timeout(in millsec)
timeout=3000

# remote=1 ... connect to MxNet1
# remote=0 ... access to the local dongle
remote=1
```

7.2 Static API

The static API libraries are compatible with VC compiler.

Configuration file for the static API library

The static libraries also read the configuration file named “mxnetapi.ini” in the same folder as the program that is linked with the remote API library for the optional values like the server IP address and the port. You can use the same Mxnetapi.ini both for DLL and static libraries.

7.3 Run-time Configuration

The following APIs will override the optional values set in the configuration file.

_mxINT32 WINAPI MxNet_SetPort(_mxINT32 _port)

Set the server socket port(“port”) to _port

_mxINT16 WINAPI rSetConfig_MatrixNet(_mxINT16 nAccess, char* nFile)

- nAccess = 1 set “remote” to 1.
  Set either the server computername or IP address to nFile
- nAccess = 0 set “remote” to 0
-

WCHAR* WINAPI MxNet_SetIP(WCHAR* _ip)

Set the server IP (“ip”) address to _ip

_mxINT32 WINAPI MxNet_SetTimeOut(_mxINT32 nTimeOut)

Set the connect timeout (“timeout”)to nTimeOut

rInit_MatrixAPI will connect to mxnet2 server using the values set by these API. Thus, you must call them before rInit_MatrixAPI.

8. Sample Programs

8.1 C Sample

In Samples¥C Sample¥testDLL folder, you find the solution file, testDLL.sln and the project folders. The output of each project will be Release or x64¥release.

The executable files, testDLL, testSt.exe, testMD.exe, will be generated, linking these different libraries（import / MT / MD）

The sample program is multi-threaded; it starts several threads. Each thread calls the same test routine repeatedly. Try to run multiple instances of the sample program.

8.2 C# Sample

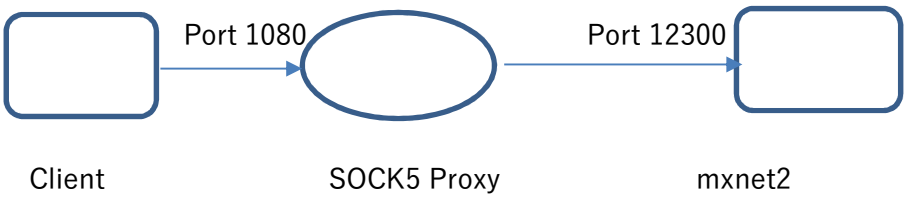
.NET program must import API from API DLL.

.NET programs built with the platform target set to “Any CPU” will run as 64bit program on 64bit Windows and 32bit programs on 32bit Windows. Since 32bit program cannot be used with 64bit DLL and vice versa, be sure to install the right DLL.

When you build a program with the platform target set to “x86”, then, the program always run as 32bit program. With “x64”, it will always run as 64bit program.

9 Using Proxy Server

The client API can connect to Mxnet2 server via SOCK5 proxy server.



In the client configuration file, create a new section [SOCKSSERVER]. Under the section, specify the IP and port of the proxy server.

[SocksServer]
Port=1080 (SOCKS5 proxy server Port)
IP=127.0.0.1 (SOCKS5 proxy server IP)

[Options]
ip=127.0.0.1
port=12300

If SOCKS5 server requires the user/password authentication, give an encrypted string value to UserPass key

[SocksServer]
Ip=(SOCKS5server IP)
Port=(SOCKS5server port)
UserPass=(encrypted user / pass)

You can generate the encrypted user/pass string using userpass.exe included in the distribution file.

Enter SOCK5 proxy server username and password and press [OK]. The encrypted string will be shown in the lower box. Copy the string and set it to “userpass” key.

Client API Error / Socks Related Errors

Error No	Description
-200	SOCKS Version error
-201	SOCKS does not support authentication
-202	SOCKS authentication error
-203	Send error
-204	Receive error
-205	Socket error(WSAEventSelect)
-206	Socket error(WSAEnumNetworkEvents)

-200 SOCKS Version error

-201 SOCKS does not support authentication

-202 SOCKS authentication error

-203 send error

-204 receive error

-205 Socket error(WSAEventSelect)

-206 Socket error(WSAEnumNetworkEvents)