LXDeviceAPI

LXDeviceAPI Developer Manual

Doc. ID. LXE64 V1

Release Date. 2017-04-25 .

Abstract –API for bio signal measuring devices

Site for LXDeviceAPI : http://laxtha.net/lxdeviceapi/

목차

OVERVIEW LXDEVICEAPI 5

FEATURES LXDEVICEAPI. 5

GETTING STARTED 6

STEP 1. DOWNLOAD / UBLOCK / UNZIP LXDEVICEAPI 6

step1.1 Download. 6

step1.2 Unblock the zip file. 6

step 1.3 unzip. 7

STEP 2 . COPY FILES TO YOUR PROJECT FOLDER 7

STEP 3.IMPORTING DLL 7

API STREAM DATA 8

STRUCT : ST_STREAMDATA_LXDAPI 8

USAGES. 8

MEMORY MAP 9

Wave_StreamData_CS Data Allocation. 9

Event_StreamData_CS Data Allocation 9

ARRAY INDEXING FOR WAVE DATA. 10

ARRAY INDEXING FOR EVENT DATA 10

INDEXING EXAMPLE. 11

API MESSAGE 12

MESSAGES BY LXDEVICEAPI 12

MESSAGE PARAMETER - WPARAM 12

MESSAGE TYPE VS. MEANING 12

CODE EXAMPLES 13

Code Example 1 . SetMessageDevice 13

Code Example 2. StartStream 13

Code Example 3. Message Handler 13

API FUNCTIONS 14

OPENAPI 15

PARAMETER 15

Return Values 15

Code Example 15

CLOSEAPI 16

Return Values 16

Code Example 16

OPENDEVICE 17

Parameter 17

Return Values 17

Code Example1 18

Code Example2 18

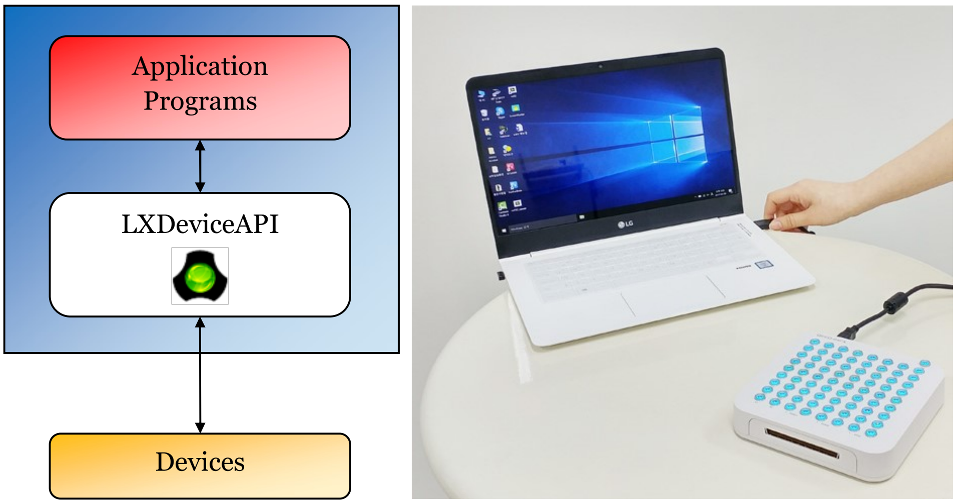
CLOSEDEVICE 19

Parameter 19

Return Values 19
Code Example 19
StartStream 20
Parameter 20
Return Values 20
Code Example 20
StoptStream 21
Parameter 21
Return Values 21
Code Example 21
GetStreamData 22
Parameter 22
Return Values 22
Code Example 22
EventMarkingOnStream 24
Parameter 24
Return Values 24
Code Example 1 24
Code Example 2 25
SetMessageDevice 26
Parameter 26
Return Values 26
Code Example 26
SetSampleFrequency 27
Parameter 27
Return Values 27
Code Example 27
SetDeviceControlPanel 28
Parameter 28
Return Values 28
Code Example 28
GetFilterFrequency 29
Parameter 29
Return Values 29
Code Example 30
GetSampleFrequency 31
Parameter 31
Return Values 31
Code Example 31
GetEEGRefElectrode 32
Parameter 32
Return Values 32
Code Example 32
CheckForUpdate 33
Parameter 33
Return Values 33
Code Example 33
APPENDIX 1. SUPPORTING DEVICES **34**
REVISION HISTORY **35**

Overview LXDeviceAPI

The devices can be communicated with your application program via LXDeviceAPI. You can make your own application programs for communicating with devices. LXDeviceAPI provides the standard C libraries and messages.



System Architecture

Features LXDeviceAPI.

- Real time bio signal data streaming from measuring device to your application program.
- Extremely stable and reliable thread for real time streaming data.
- Setting the device parameters (Sampling Frequency, Device Mode Change)
- Providing the self contained UI.
  - Setting the signal filter(LPF, HPF, notch) via “device control panel”.
  - Electrode-Skin Impedance Monitoring.
  - Auto Calibrating the device.
  - Auto Self Updating API.
  - Auto Device Firmware Update.
  - Saving the configuration information into Device.
- API type : DLL (Dynamic Link Library).
- API Making tool : Visual C++ 2015. MFC Regular DLL project.
- Supporting platform : both 32bit and 64bit.
- Supprting OS : Windows 10, 8.1, 8, 7.

Getting Started

Step 1. Download / Ublock / Unzip LXDeviceAPI

step1.1 Download.

Click below link,

LXDeviceAPI_32bit.zip for 32 bit Applications.

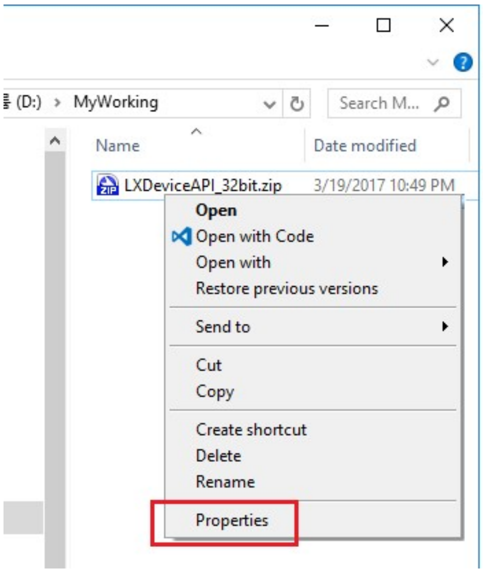
https://github.com/LAXTHA/LXDeviceAPI/raw/master/LXDeviceAPI_32bit.zip

LXDeviceAPI_64bit.zip for 64bit Applications.

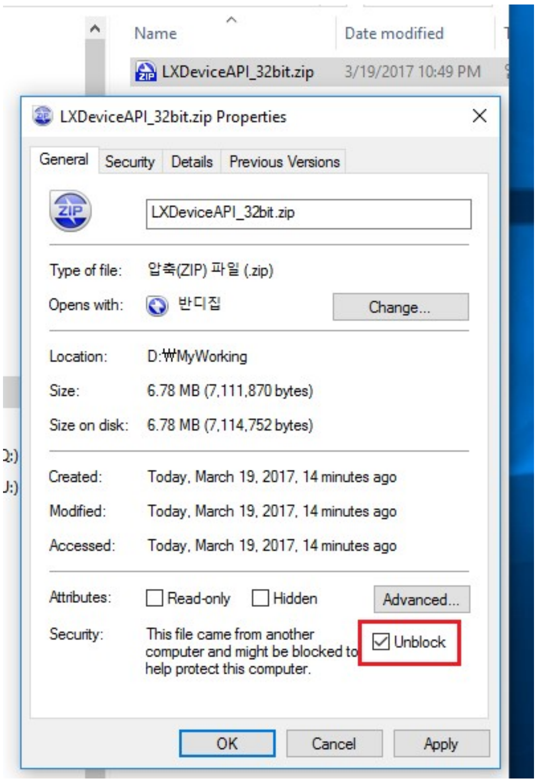
https://github.com/LAXTHA/LXDeviceAPI/raw/master/LXDeviceAPI_64bit.zip

step1.2 Unblock the zip file..

Before unzip, unblock the zip file should be taken. Click the downloaded zip file by right mouse and click the Properties (red box following left image). Check Unblock and then click OK.



->



LXDeviceAPI_32bit.zip

Type of file: 압축(ZIP) 파일 (.zip)

Opens with: 반디집

Location: D:₩MyWorking

Size: 6.78 MB (7,111,870 bytes)

Size on disk: 6.78 MB (7,114,752 bytes)

Created: Today, March 19, 2017, 14 minutes ago

Modified: Today, March 19, 2017, 14 minutes ago

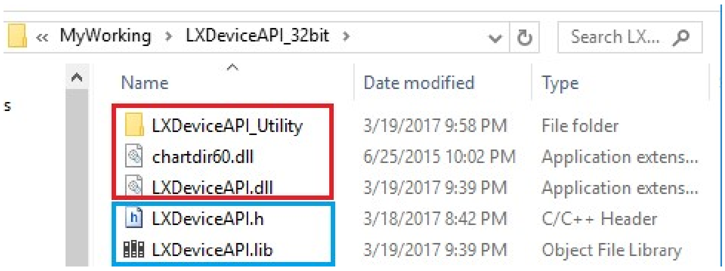
Accessed: Today, March 19, 2017, 14 minutes ago

Attributes: □ Read-only □ Hidden

Security: This file came from another computer and might be blocked to help protect this computer. ☑ Unblock

step 1.3 unzip.

After unzip, you can see the files and folder as following.



« MyWorking › LXDeviceAPI_32bit ›

Search LX...

Name	Date modified	Type
LXDeviceAPI_Utility	3/19/2017 9:58 PM	File folder
chartdir60.dll	6/25/2015 10:02 PM	Application extens...
LXDeviceAPI.dll	3/19/2017 9:39 PM	Application extens...
LXDeviceAPI.h	3/18/2017 8:42 PM	C/C++ Header
LXDeviceAPI.lib	3/19/2017 9:39 PM	Object File Library

Step 2 . Copy files to your project folder

Files	Copy to
● LXDeviceAPI_Utility (폴더) ● LXDeviceAPI.DLL ● chartdir60.dll	copy to the same folder which contains your application exe file.
● LXDeviceAPI.LIB ● LXDeviceAPI.h	copy to your application project's source folder.

Step 3.Importing DLL

In your Visual C++ source file, add the codes for implicit linking the library and include the LXDeviceAPI.h

```
#pragma comment(lib,"LXDeviceAPI.lib") // implicit linking.

#include "LXDeviceAPI.h" // for using LXDeviceAPI
```

That's all, now it's ready to use all functions and message from LXDeviceAPI.

API Stream Data

Struct : ST_STREAMDATA_LXDAPI

LXDeviceAPI.h provides the typedef of ST_STREAMDATA_LXDAPI.

```
// struct for stream.
typedef struct _stStreamData_LXDAPI
{
…
            unsigned int*           Event_StreamData_CS; // array for event stream
…
            double*                 Wave_StreamData_CS; //array for measured multi channel bio signal

…
}ST_STREAMDATA_LXDAPI;
```

The most important member variables are Wave_StreamData_CS for multi channel bio signal and Event_StreamData_CS for event marking.

double * Wave_StreamData_CS

unsigned int* Event_StreamData_CS

Usages.

Declare the variable for stream data type and feed into OpenDevice as a address of variable.

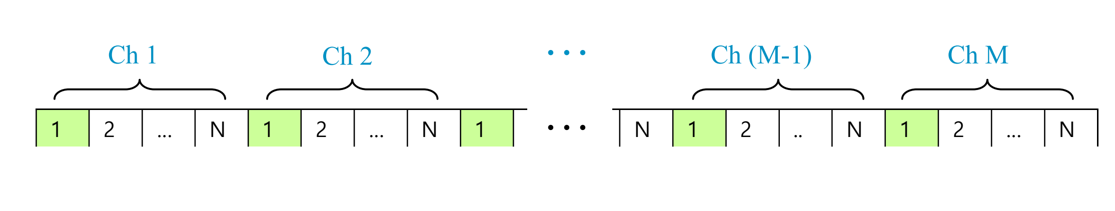
```
ST_STREAMDATA_LXDAPI  stStreamData;    // .
int NumSampleReturn = 32 ; // Available Value 위 : 1 ~ 128
OpenDevice_LXDeviceAPI( , &stStreamData, NumSampleReturn, );  //
```

LXDeviceAPI dynamic allocates the memory for arrays when you call the OpenDevice. The size of memory is determined by the OpenDevice's parameter int numsample_return which is the number of samples per each message received. LXDeviceAPI deletes the memory if CloseDevice is called.

Memory Map.

Wave_StreamData_CS Data Allocation.

Wave Stream Data. Memory Map & Array Indexing.

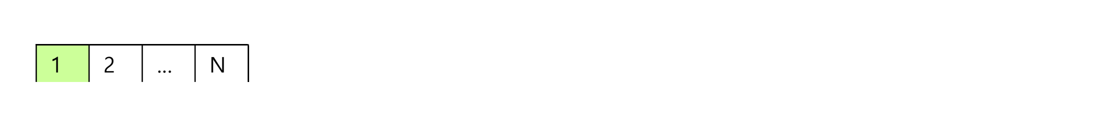


Wave_StreamData_CS [sample_index + N * channel_index]

- **N** : Number of Samples
- **M** : Number of Channels
- **sample_index :** index for samples. from 0 to N-1
- **channel_index** : index for channels. from 0 to M-1

Event_StreamData_CS Data Allocation

Event Stream Data. Memory Map & Array Indexing.



Event_StreamData_CS [sample_index]

- **N** : Number of Samples
- **sample_index :** index for samples. from 0 to N-1

Array Indexing for Wave data

The following code shows how to get the one sample for specific channel index and sample index,

```
double one_sample_wave = stStreamData.Wave_StreamData_CS[sample_index + NumSampleReturn *
channel_index];
```

where,

channel_index : Available value range form 0 to (number of channel – 1). The number of channel is specific to device. In the case of QEEG-64FX, the number of channel is 67 (EEG 64ch + Bipolar 3ch).

sample_index : Available value range from 0 to (NumSampleReturn – 1). The NumSampleReturn is determined by the parameter int numsample_return when you call the OpenDevice.

Array Indexing for Event Data

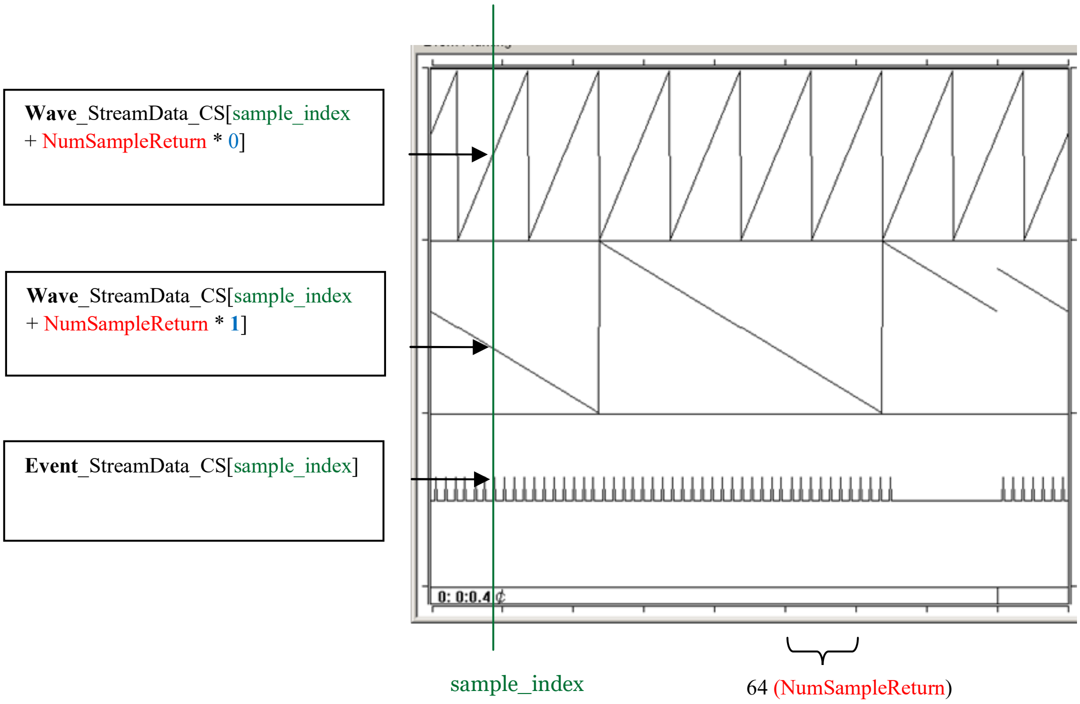
```
unsigned int one_sample_event = stStreamData.Event_StreamData_CS[sample_index];
```

where,

sample_index : Available value range from 0 to (NumSampleReturn- 1).The NumSampleReturn is determined by the parameter int numsample_return when you call the OpenDevice.

Indexing Example

Array indexing example for two channel signals and event. The chart plots the both signals and event simultaneously.



API Message

Messages by LXDeviceAPI

LXDeviceAPI send mesage to application program using win32 function SendMessage(.,.,wParam, lParam) . The message parameter wParam has informations to recognize the message type and what processing should be taken by application program.

Message Parameter - wParam

LXDeviceAPI send a wParam as an unsigned int type(byte size 4). Application program can take the information at each byte as follows; wParam is divided into 4 bytes. Byte0 is lowest byte and Byte3 means highest byte.

Parameter	Written Data	Description.
wParam_Byte3	Device Handling ID	● Device Handling ID = wParam_Byte3*256 + wParam_Byte2. range : 1~65535 ● The message is sent by device correspond to "Device Handling ID". ● Related : Device Handling ID can be retrived by return value of OpenDevice.
wParam_Byte2		
wParam_Byte1	Message Type ID.	● Identificating the message. ● available values : MSGTYPEID#_DEVICE_LXDAPI, MSGTYPEID#_API_LXDAPI predefined on LXDeviceAPI.h. ● Related Function : SetMessageDevice
wParam_Byte0	Message Type Sub ID.	● sub message type under Message Type ID.

Message Type vs. Meaning

wParam Byte 1 (Message Type ID)	**wParam Byte 0** (Message Type Sub ID)	**Description**
MSGTYPEID0_DEVICE_LXDAPI	0	The number of sampling data is ready. App's message handler must call the function GetStreamData to get the stream data from LXDeviceAPI. The "number of sampling" is defined by int numsample_return when calling the function OpenDevice.

Code Examples

Code Example 1 . SetMessageDevice

```
#define WM_STREAM_DEVICE WM_USER+203  // Define Message to get message from LXDeviceAPI.

void CLXDeviceAPI_Sample1View::OnMenuSetmessagedevice()
{
        SetMessageDevice_LXDeviceAPI(m_iDeviceHandlingID, MSGTYPEID0_DEVICE_LXDAPI ,this->m_hWnd,
WM_STREAM_DEVICE,1); //
}
```

Code Example 2. StartStream

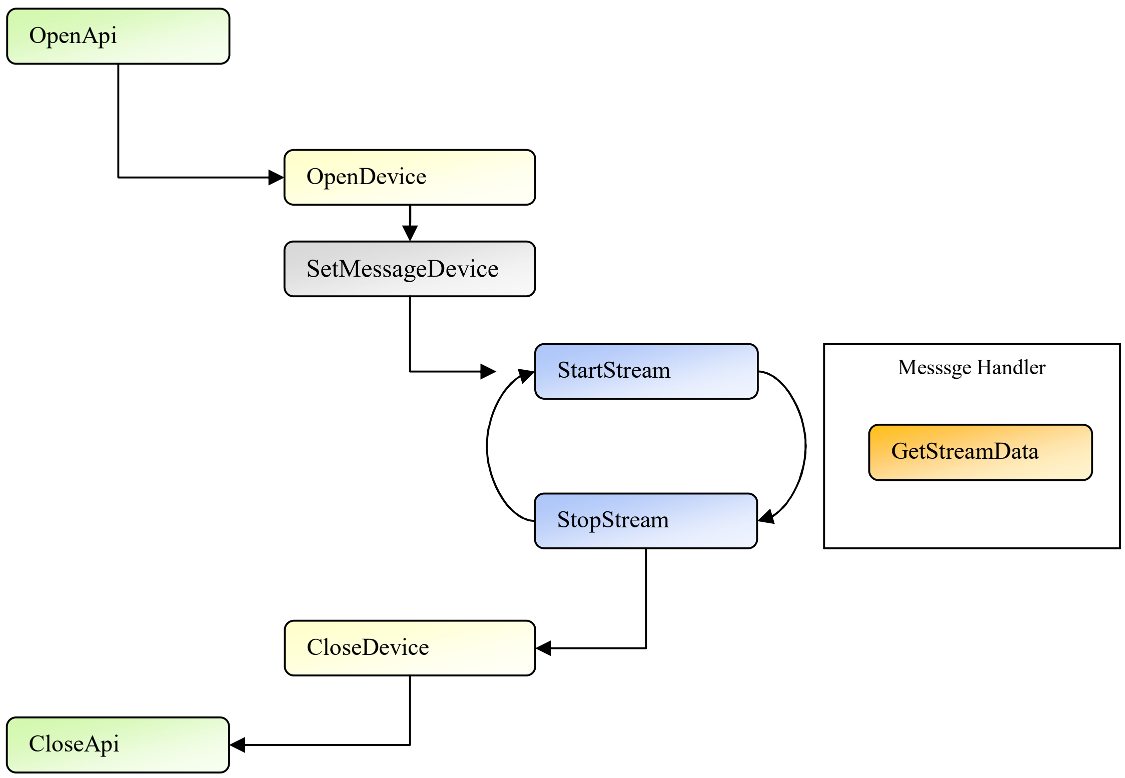
```
void CLXDeviceAPI_Sample1View::OnMenuStartstream()
{

        StartStream_LXDeviceAPI(m_iDeviceHandlingID);
}
```

Code Example 3. Message Handler

After calling the function StartStream, the LXDeviceAPI sends a message when the number of sample data is ready. Your app's message handler must call the function GetStreamData to get the stream data from LXDeviceAPI. The "number of sampling" is defined by int numsample_return when calling the function OpenDevice. If you stop the message from LXDeviceAPI, call the function StopStream or call the SetMessageDevice as last parameter = 0.

```

/* Real Time Acquisition */

afx_msg LRESULT CLXDeviceAPI_Sample1View::OnStreamData(WPARAM wParam, LPARAM lParam)
{
        unsigned int uintWPARAM = (unsigned int)wParam;

        unsigned char msgtype_id = (unsigned char)(uintWPARAM >> 8); //get the lowest 2'nd byte(message type id).

        unsigned char msgtype_subid = (unsigned char)(uintWPARAM); //get the lowest 1st byte(message type sub id).

        switch (msgtype_id)
        {
        case MSGTYPEID0_DEVICE_LXDAPI: // for real time stream type messages.

                switch (msgtype_subid)
                {
                case 0:
                        GetStreamData_LXDeviceAPI(uintWPARAM); //  the new stream data is allocated on stStreamData
which is ST_STREAMDATA_LXDAPI type variable.
                       // At this point, you can use stStreamData
                        break;
                } // switch (msgtype_subid)
                break;  // case MSGTYPEID0_DEVICE_LXDAPI: // for real time stream type messages.
        }// switch (msgtype_id)

        return 0;
}
```

API Functions



OpenApi

int OpenApi_LXDeviceAPI(int api_window, int api_selfupdate, int mode)

Parameter

Parameter	Available Value.	Description
int api_window	0 : "api window" view off. 1 : "api window" view on.(default)	api window view enable/disable.
int api_selfupdate	0 : "api check self update" execution. 1 ; "api check self update" no exec.(default)	api self-upadte check execution or not.
int mode	0 ☹default)	Reserved

Return Values

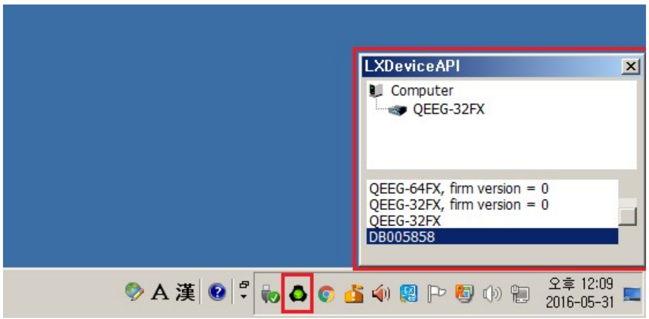
Return	Meaning	Description
1	Success	
-1	Fail	Already succefully called OpenApi.
-2	Fail	Windows 8/10. LXDeviceAPI fixing the os registry for USB normal communication.
-5	Fail. Duplicated call	When OpenApi called, LXDeviceAPI loading the internal used DLL from the folder LXDeviceAPI_Utility.

Code Example

```
void CLXDeviceAPI_Sample1View::OnMenuOpenapi()
{
            OpenApi_LXDeviceAPI(1,0,0);
}
```

Result

If OpenApi is called successfully, LXDeviceAPI window & tray icon appears at the bottom right.



```
QEEG-64FX, firm version = 0
QEEG-32FX, firm version = 0
QEEG-32FX
DB005858
```

LXDeviceAPI Window Enabled.

CloseApi

```
int CloseApi_LXDeviceAPI()
```

Return Values

Return Values	Meaning	Description
1	Success	
-10	Fail. Nothing to close	CloseApi should be called after OpenApi.

Code Example

```
void CLXDeviceAPI_Sample1View::OnMenuCloseapi()
{
                CloseApi_LXDeviceAPI();
}
```

OpenDevice

int OpenDevice_LXDeviceAPI(int LXDeviceID,ST_STREAMDATA_LXDAPI* p_streamdata,int numsample_return,int mode)

Parameter

Parameter	Available Value	Description
int LXDeviceID	1~65535	Uinque ID of the device to communicate.

ST_STREAMDATA_LXDAPI* p_stream_data	Address struct type of ST_STREAMDATA_LXDAPI	example.
		ST_STREAMDATA_LXDAPI stStreamData;
		OpenDevice_LXDeviceAPI(, &stStreamData, ,);

int numsample_return	1~128	The number of sampling point per one stream message.
int mode	0: (default)	reserved

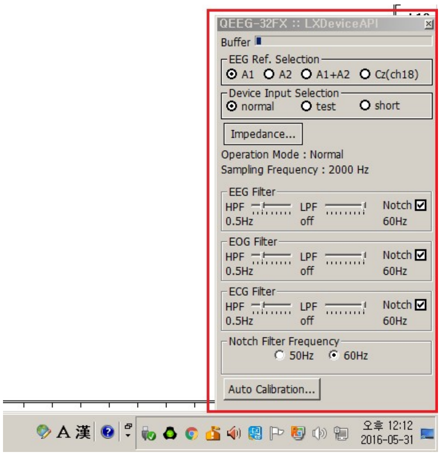
Return Values

Return Values	Meaning	Description
>0	Success. Device Handling ID.	Application program must save the return value to call the other functions which has a parameter int device_handling_id
-1	Fail. Not supporting device.	
-2	Fail. Duplicated Call	Already succefully called OpenApi.
-3	Fail. No device found matching LXDeviceID.	
-5	Fail. Need to update the device firmware.	Try again after completing the device firmware update.
-10	Fail. Wrong Calling order	OpenDevice should be called after OpenApi.
-20	Fail. Wrong paramter	int numsample_return should be range of 1~128.

Code Example1

```
void CLXDeviceAPI_Sample1View::OnMenuOpendevice()
{
		int retv = OpenDevice_LXDeviceAPI(300, &stStreamData, NumSampleReturn, 0);// QEEG-32FX(LXDeviceID=300) Open

		if(retv > 0) // if success
		{
				m_iDeviceHandlingID = retv;
		}
		else
		{
				AfxMessageBox(_T("Fail to OpenDevice"));
		}
}
```

Result .



QEEG-32FX :: LXDeviceAPI

Buffer

EEG Ref. Selection
● A1 ○ A2 ○ A1+A2 ○ Cz(ch18)

Device Input Selection
● normal ○ test ○ short

Impedance...

Operation Mode : Normal
Sampling Frequency : 2000 Hz

EEG Filter
HPF 0.5Hz LPF off Notch ☑ 60Hz

EOG Filter
HPF 0.5Hz LPF off Notch ☑ 60Hz

ECG Filter
HPF 0.5Hz LPF off Notch ☑ 60Hz

Notch Filter Frequency
○ 50Hz ● 60Hz

Auto Calibration...

Device Control Panel Enabled.

Code Example2

```
/*
OpenAPI-> OpenDevice -> StartStream  .
*/
void CLXDeviceAPI_Sample1View::OnMenuOneclickstart()
{
		int retv_openapi, retv_opendevice;

		retv_openapi = OpenApi_LXDeviceAPI(1, 0, 0); // API Open.

		if (retv_openapi > 0)
		{
				retv_opendevice = OpenDevice_LXDeviceAPI(300, &stStreamData, NumSampleReturn, 0); // QEEG-32FX Open
				if (retv_opendevice > 0)
				{
						m_iDeviceHandlingID = retv_opendevice;
						SetMessageDevice_LXDeviceAPI(m_iDeviceHandlingID, 0, this->m_hWnd,
WM_LXDAPI_MSGTYPEID_0, 1);

						StartStream_LXDeviceAPI(m_iDeviceHandlingID);
				}
		}
}
```

CloseDevice

```
int CloseDevice_LXDeviceAPI(int device_handling_id);
```

Parameter

Parameter	Available Value.	Description
int device_handling_id	return value of OpenDevice	

Return Values

Return Values	Meaning	Description
1	Success	
-3	Fail. No device matching device_handling_id.	
-10	Fail. Wrong calling order.	CloseDevice should be called after OpenApi.

Code Example

```
void CLXDeviceAPI_Sample1View::OnMenuClosedevice()
{
		CloseDevice_LXDeviceAPI(m_iDeviceHandlingID);
}
```

StartStream

int StartStream_LXDeviceAPI(int device_handling_id, int mode)

Parameter

Parameter	Available Value.	Description
int device_handling_id	return value of OpenDevice	
int mode	0 : (Default)	Reserved.

Return Values

Return Values	Meaning	Description
1	Success	
-3	Fail	No device(correspond to int device_handling_id).
-5	Fail	Impedance measuring mode.
-6	Fail	Auto calibrarion mode.
-10	Fail	Wrong calling order. StartStream should be called after OpenApi.

Code Example

```
void CLXDeviceAPI_Sample1View::OnMenuStartstream()
{

		StartStream_LXDeviceAPI(m_iDeviceHandlingID);
}
```

StoptStream

```
int StopStream_ LXDeviceAPI (int device_handling_id);
```

Parameter

Parameter	Available Value.	Description
int device_handling_id	return value of OpenDevice	

Return Values

Return Values	Meaning	Description
1	Seccess	
-3	Fail	No device(correspond to int device_handling_id).
-10	Fail	Wrong calling order. StopStream should be called after OpenApi.

Code Example

```
void CLXDeviceAPI_Sample1View::OnMenuStopstream()
{
		StopStream_LXDeviceAPI(m_iDeviceHandlingID);
}
```

GetStreamData

int GetStreamData_LXDeviceAPI(unsigned int message_wparam);

Parameter

Parameter	Available Value.	Description
unsigned int message_wparam	(unsigned int) wParam.	The parameter wParam from application's message handler function.

Return Values

Return Values	Meaning	Description
1	Success	
-3	Fail	No device(correspond to int device_handling_id).
-10	Fail	Wrong calling order. GetStreamData should be called after OpenApi.

Code Example

```
afx_msg LRESULT CLXDeviceAPI_Sample1View::OnStreamData(WPARAM wParam, LPARAM lParam)
{
          unsigned int uintWPARAM = (unsigned int)wParam;

          unsigned char msgtype_id = (unsigned char)(uintWPARAM >> 8); //get the lowest 2'nd byte(message
type id).

          unsigned char msgtype_subid = (unsigned char)(uintWPARAM); //get the lowest 1st byte(message type
sub id).

          switch (msgtype_id)
          {
          case MSGTYPEID0_DEVICE_LXDAPI: // for real time stream type messages.

                    switch (msgtype_subid)
                    {
                    case 0:

                              GetStreamData_LXDeviceAPI(uintWPARAM); //  the new stream data is allocated on
stStreamData which is ST_STREAMDATA_LXDAPI type variable.

                              /// Data arrange for Wave Stream Data for plotting.
                              for (int i = 0; i < NumSampleReturn * NumChannel_Wave; i++)
                              {
                                        testfloat_Wave[i] = (float)stStreamData.Wave_StreamData_CS[i];
                              }


                              /// Data arrange for Event Marking for plotting.
                              for (int idx_event = 0; idx_event < NumSampleReturn; idx_event++)
                                        testfloat_Wave[NumSampleReturn * NumChannel_Wave + idx_event] =
(float)stStreamData.Event_StreamData_CS[idx_event];

                              // Test plotting the wave and event marking.
                              ACQPLOT_DLL_Array_Datain_Strip(testfloat_Wave, NumChannel_Wave + 1,
NumSampleReturn); // parameter 1 : wave data float type array. parameter 2 : total number of wave channel  +
1 for Event Marking. parameter 3 : Number of Samples per one channel.


                              break;
```

```
                } // switch (msgtype_subid)

                break;  // case MSGTYPEID0_DEVICE_LXDAPI: // for real time stream type messages.
        }// switch (msgtype_id)


        return 0;
}
```

EventMarkingOnStream

int EventMarkingOnStream_LXDeviceAPI(int device_handling_id, unsigned int event_id);

Parameter

Parameter	Available Value.	Description
int device_handling_id	return value of OpenDevice	
unsigned int event_id	1~ 65535.	The event_id will be marked on signal stream. When you retrieve the marked value from stream, you can recognize the what type of the event(press the key , image shown, etc.) has been marked on stream.

Return Values

Return Values	Meaning	Description
1	Success	
-3	Fail	No device(correspond to int device_handling_id).
-10	Fail	Wrong calling order. Should be called after OpenApi.

Code Example 1

```
// when menu clicked, let's set the event_id as 20000
void CLXDeviceAPI_Sample1View::OnMenuEventmarkingonstream()
{
	EventMarkingOnStream_LXDeviceAPI(m_iDeviceHandlingID, 20000);
}
```

Code Example 2

```
// when keyboard pressed. let's set the event_id as arrow ky down. up 30000, down 40000,left 50000, right
60000

void CLXDeviceAPI_Sample1View::OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)
{

        switch (nChar)
        {
        case VK_UP:

                EventMarkingOnStream_LXDeviceAPI(m_iDeviceHandlingID, 30000);

                break;

        case VK_DOWN:

                EventMarkingOnStream_LXDeviceAPI(m_iDeviceHandlingID, 40000);

                break;

        case VK_LEFT:

                EventMarkingOnStream_LXDeviceAPI(m_iDeviceHandlingID, 50000);

                break;

        case VK_RIGHT:

                EventMarkingOnStream_LXDeviceAPI(m_iDeviceHandlingID, 60000);

                break;

        }

        // TODO: Add your message handler code here and/or call default

        CView::OnKeyDown(nChar, nRepCnt, nFlags);
}
```

SetMessageDevice

int SetMessageDevice_LXDeviceAPI(int device_handling_id,int msgtype_id,HWND hwnd_msgrcv,int msg_id,int onoff);

Parameter

Parameter	Available Value.	Description
int device_handling_id	return value of OpenDevice	
int msgtype_id	MSGTYPEID0_DEVICE_LXDAPI ... MSGTYPEID9_DEVICE_LXDAPI	Predefined on LXDeviceAPI.h
HWND hwnd_msgrcv	window handle	App.'s window handle to receive the message from LXDeviceAPI.
int msg_id	WM_USER ~ WM_USER+31643 WM_APP ~ WM_APP+16383	You should define the unique message id without duplication with the other messages in your project.
int onoff	0 : message off 1 : message on	message(correspond to int msgtype_id) on/off

Return Values

Return Values	Meaning	Description
1	Success	
-3	Fail	No device(correspond to int device_handling_id).
-10	Fail	Wrong calling order. SetMessageDevice should be called after OpenApi.

Code Example

```
#define WM_STREAM_DEVICE WM_USER+203  // Define Message to get message from LXDeviceAPI.

void CLXDeviceAPI_Sample1View::OnMenuSetmessagedevice()
{
          SetMessageDevice_LXDeviceAPI(m_iDeviceHandlingID, MSGTYPEID0_DEVICE_LXDAPI ,this->m_hWnd,
WM_STREAM_DEVICE,1); //
}
```

SetSampleFrequency

int SetSampleFrequency_LXDeviceAPI(int device_handling_id, unsigned int sample_frequency)

Parameter

Parameter	Available Value.	Description
int device_handling_id	return value of OpenDevice	
unsigned int sample_frequency	250,500,1000,2000	Sampling Frequency. unit : Hz.

Return Values

Return Values	Meaning	Description
1	Success	
-3	Fail	No device(correspond to int device_handling_id).
-10	Fail	Wrong calling order. SetMessageDevice should be called after OpenApi.
-20	Fail	Not supporting value of unsigned int sample_frequency

Code Example

```
/*
기기의 샘플링 주파수 2000Hz로 설정.
*/
void CLXDeviceAPI_Sample1View::OnSetsamplefrequency2000hz()
{
		SetSampleFrequency_LXDeviceAPI(m_iDeviceHandlingID, 2000);
}
```

SetDeviceControlPanel

```
int SetDeviceControlPanel_LXDeviceAPI(int device_handling_id,int para0, int para1);
```

Parameter

Parameter	Available Value.	Description
int device_handling_id	return value of OpenDevice	
int para0	0 : Select all elements.	Target elements in API's Device Control Panael.
int para1	0 : disable 1 : enable	Enable/disable the target elements selected by int para0

Return Values

Return Values	Meaning	Description
1 이상	Success	
-3	Fail	No device(correspond to int device_handling_id).
-10	Fail	Wrong calling order. StopStream should be called after OpenApi.

Code Example

```
void CLXDeviceAPI_Sample1View::OnMenuSetdevicecontrolpanel()
{
            static int para1=0;
            SetDeviceControlPanel_LXDeviceAPI(m_iDeviceHandlingID, 0, para1);
            // toggling para1.
            if (para1) para1 = 0;
            else para1 = 1;
}
```

GetFilterFrequency

int GetFilterFrequency_LXDeviceAPI(int device_handling_id,int signal_source, float * freq_hpf, float * freq_lpf, float * freq_notch);

Parameter

Parameter	Available Value.	Description
int device_handling_id	return value of OpenDevice	
int signal_source	0 : EEG 1 : EOG 2 : ECG	Select bio signal type to retrieve the applied filter information.
float * freq_hpf	output	High Pass Filter cut frequency. unit : Hz. The value -100 means No HPF applied.
float * freq_lpf	output	Low Pass Filter cut frequency. unit : Hz. The value -100 means no LPF applied.
float * freq_notch	output	Notch Filter frequency. uint : Hz. The value -100 means no notch filter applied.

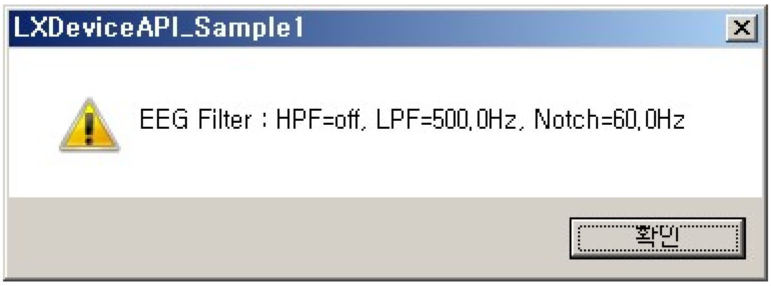
Return Values

Return Values	Meaning	Description
1	Success	
-3	Fail	No device(correspond to int device_handling_id).
-10	Fail	Wrong calling order. GetFilterFrequency should be called after OpenApi.

Code Example

```
void CLXDeviceAPI_Sample1View::OnMenuGetfilterfrequency()
{
        float f_hpf, f_lpf, f_notch;
        CString cst_hpf, cst_lpf, cst_notch;
        // Get EEG Filter info
        if (GetFilterFrequency_LXDeviceAPI(m_iDeviceHandlingID,0, &f_hpf, &f_lpf, &f_notch) == 1)
        {
                if (f_hpf < -1.f)       cst_hpf = _T("HPF=off, ");
                else                    cst_hpf.Format(_T("HPF=%.1fHz, "), f_hpf);

                if (f_lpf < -1.f)       cst_lpf = _T("LPF=off, ");
                else                    cst_lpf.Format(_T("LPF=%.1fHz, "), f_lpf);

                if (f_notch < -1.f)     cst_notch = _T("Notch=off");
                else                    cst_notch.Format(_T("Notch=%.1fHz"), f_notch);

                AfxMessageBox(_T("EEG Filter : ") + cst_hpf + cst_lpf + cst_notch);
        }
}
```

Result



EEG Filter : HPF=off, LPF=500.0Hz, Notch=60.0Hz

GetSampleFrequency

```
int GetSampleFrequency_LXDeviceAPI(int device_handling_id, int * sample_frequency);
```

Parameter

Parameter	Available Value.	Description
int device_handling_id	return value of OpenDevice	
int * sample_frequency	output	

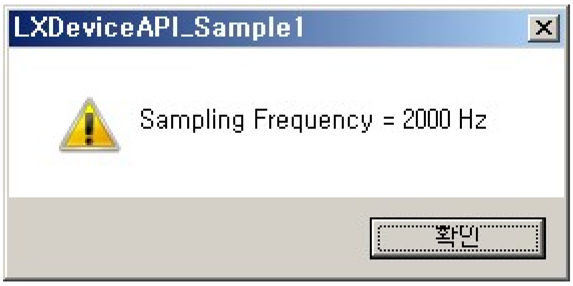
Return Values

Return Values	Meaning	Description
1	Success	
-3	Fail	No device(correspond to int device_handling_id).
-10	Fail	Wrong calling order. GetFilterFrequency should be called after OpenApi.

Code Example

```
void CLXDeviceAPI_Sample1View::OnMenuGetsamplefrequency()
{
		int sample_frequency;
		CString cst;

		if (GetSampleFrequency_LXDeviceAPI(m_iDeviceHandlingID, &sample_frequency) == 1)
		{
			cst.Format(_T("Sampling Frequency = %d Hz"), sample_frequency);

			AfxMessageBox(cst);
		}
}
```

Result



LXDeviceAPI_Sample1

Sampling Frequency = 2000 Hz

GetEEGRefElectrode

int GetEEGRefElectrode_LXDeviceAPI(int device_handling_id , int * eeg_refelectrode);

Parameter

Parameter	Available Value.	Description
int device_handling_id	return value of OpenDevice	
int * eeg_refelectrode	0 : A1 1 : A2 2 : A1,A2 3 : Cz (ch18)	Retrieving the selected EEG reference electrode. Cz is available only if EEG Cap is introduced.

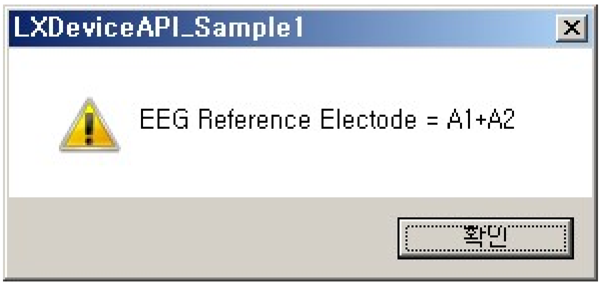
Return Values

Return Values	Meaning	Description
1	Success	
-3	Fail	No device(correspond to int device_handling_id).
-10	Fail	Wrong calling order. GetFilterFrequency should be called after OpenApi.

Code Example

```
void CLXDeviceAPI_Sample1View::OnMenuGeteegrefelectrode()
{
                int eeg_refelectrode;
                CString cst_eegref;

                if (GetEEGRefElectrode_LXDeviceAPI(m_iDeviceHandlingID,&eeg_refelectrode) == 1)
                {
                                if (eeg_refelectrode == 0)                              cst_eegref = _T("A1");
                                else if(eeg_refelectrode == 1)                          cst_eegref = _T("A2");
                                else if (eeg_refelectrode == 2)                         cst_eegref = _T("A1+A2");
                                else if (eeg_refelectrode == 3)                         cst_eegref = _T("Cz(ch18)");
                                AfxMessageBox(_T("EEG Reference Electode = ") + cst_eegref);
                }
}
```

Result



LXDeviceAPI_Sample1

EEG Reference Electode = A1+A2

CheckForUpdate

int CheckForUpdate_LXDeviceAPI(int closeifnoupdate);

Parameter

Parameter	Available Value.	Description
int closeifnoupdate	0, 1	1 : Auto close update window if there is no update 0 : Don't close update window although there is no update.

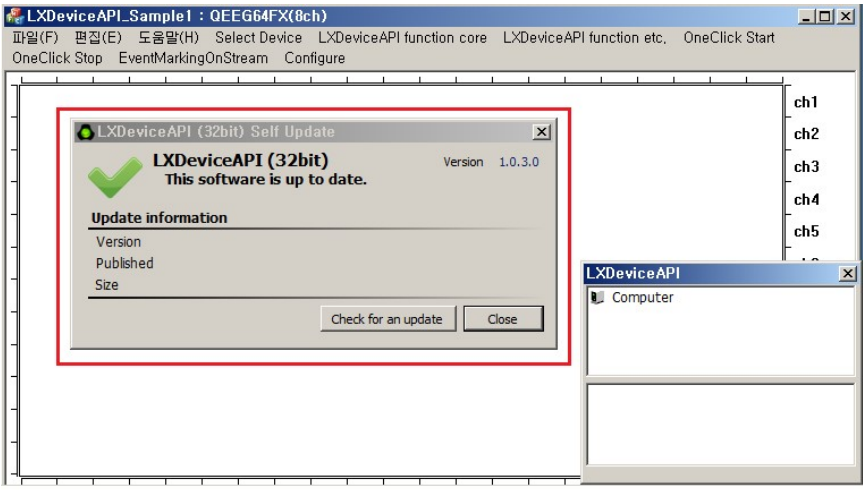
Return Values

Return Values	Meaning	Description
1 이상	Success	
-10	Fail	Wrong calling order. Should be called after OpenApi.

Code Example

```
void CLXDeviceAPI_Sample1View::OnMenuCheckforupdate()
{
        CheckForUpdate_LXDeviceAPI(0); // parameter 0 : No Auto Close
        //CheckForUpdate_LXDeviceAPI(1); // parameter 1 : Auto close check program if there is no update.
}
```

Result



LXDeviceAPI (32bit) Self Update

LXDeviceAPI (32bit)
This software is up to date.

Version 1.0.3.0

Update information
Version
Published
Size

Appendix 1. Supporting Devices .

Device Model	LXDeviceID	Sampling Frequency (Hz)	Channel index vs. signal source
QEEG-32FX	300	250Hz, 500Hz, 1000Hz, 2000Hz	0~31 : EEG channel 1~32, 32:EOG1, 33:EOG2, 34:ECG
QEEG-64FX(8ch)	16408	250Hz, 500Hz, 1000Hz, 2000Hz	0~7 : EEG channel 1~8, 8:EOG1, 9:EOG2, 10:ECG
QEEG-64FX(16ch)	16416	250Hz, 500Hz, 1000Hz, 2000Hz	0~15 : EEG channel 1~16, 16:EOG1, 17:EOG2, 18:ECG
QEEG-64FX(24ch)	16424	250Hz, 500Hz, 1000Hz, 2000Hz	0~23 : EEG channel 1~24, 24:EOG1, 25:EOG2, 26:ECG
QEEG-64FX(32ch)	16432	250Hz, 500Hz, 1000Hz, 2000Hz	0~31 : EEG channel1~32, 32:EOG1, 33:EOG2, 34:ECG
QEEG-64FX(40ch)	16440	250Hz, 500Hz, 1000Hz,	0~39 : EEG channel 1~40, 40:EOG1, 41:EOG2, 42:ECG
QEEG-64FX(48ch)	16448	250Hz, 500Hz, 1000Hz,	0~47 : EEG channel 1~48, 48:EOG1, 49:EOG2, 50:ECG
QEEG-64FX(56ch)	16456	250Hz, 500Hz, 1000Hz,	0~55 : EEG channel 1~56, 56:EOG1, 57:EOG2, 58:ECG
QEEG-64FX(64ch)	16464	250Hz, 500Hz, 1000Hz,	0~63 : EEG channel 1~64, 64:EOG1, 65:EOG2, 66:ECG

Revision History

Release Date	Doc. ID	Description of Change
2017-04-25	LXD64 V1	First release