

NKIO\_ API user manual

Version: 4.1.9

Date: 21/08/25

Author: EA



# NKIO\_ API user manual

---

NODKA

4.1.9

**2021-8-25**

Distribution list:

Name / Group	Company
EA	NODKA

Reviews/Approvals:

	Name / Function / Company	Signature
Author:	EA	
Reviewed by:		

## Table of Contents:

1	History .....	3
2	Introduction .....	4
3	Installation .....	5
4	API usage .....	6
4.1	API function list .....	6
4.2	Library Base .....	7
4.2.1	DIOLC_LibraryBaseInit .....	7
4.2.2	DIOLC_OpenDevice .....	7
4.2.3	DIOLC_CloseDevice .....	9
4.2.4	DIOLC_IsDeviceOpened .....	10
4.2.5	DIOLC_Process .....	10
4.2.6	DIOLC_LibraryBaseDeinit .....	11
4.2.7	DIOLC_SetGeneralParam .....	11
4.2.8	DIOLC_GetGeneralParam .....	14
4.3	DIO function .....	17
4.3.1	Polling Mode .....	17
4.4	Light Control function .....	20
5	Test Tool .....	29
5.1	IO Test .....	29
5.2	Light Control Test .....	30
5.2.1	Advanced .....	31
5.3	Firmware update .....	31
6	Development .....	33
7	FAQ .....	34
7.1	What's the serial communication configure parameters? .....	34
7.2	How to access the DIO channels in the multi-thread? .....	34
7.3	How to check the hardware version? .....	34
7.4	How to get the configure profile for different IO board? .....	34

## 1 History

Version	Date	Author	Description
1.0.0	2020-7-10	EA	Initial version
4.0.0	2020-12-16	EA	For hardware version 4.0
4.0.1	2020-12-21	EA	Fix some bug, add the C# example
4.1.0	2021-01-28	EA	Optimize the API to be compatible with hardware version 2.x
4.1.5	2021-04-13	EA	Add the API description for the holding time unit of the light control which is only available for the firmware version from 4.2.3.
4.1.6	2021-04-18	EA	Change DIO function description: for the DO, set 0 to turn on and set 1 to turn off.
4.1.7	2021-05-12	EA	Add the APIs to read back the digital output by bit, byte and word in the polling mode. See <a href="#">Polling Mode</a> .
4.1.8	2021-07-27	EA	Add the light control mode contents. See <a href="#">LC_SetPwmParams</a>
4.1.9	2021-08-20	EA	Add the API to turn on/off the light without save. See <a href="#">LC_SetPwmParamsNoRetentive</a>

## 2 Introduction

Nodka NP-61xx Automation PC series products provide the interface to extend the functionalities by the add-on board. Among them, NP-6111/6122-JH2 and NP-6122-H1 provide the features of PoE interface, 8 channels isolated DI, 8 channels isolated DO and 4 channels of the Pwm light control functionalities. The below tables show the IO features on different add-on boards.

Features	POE	DI/DO	Light Control
NP-6111-JH2	2	8 DI, 8 DO	4
NP-6111-JH3		8DI, 8DO	
NP-6122-H1	4	8DI, 8DO	4
NP-6122-H1B	4	16DI, 16DO	
NP-6122-JH2	2	8DI, 8DO	4
NP-6122-JH3		8DI, 8DO	
NP-6123-MVS		8DI, 8DO	4
TPC6000-XXX4		8DI, 8DO	

For the DIO and the PWM light control features, Nodka provides the dynamic library and export the function interfaces to be called by the user's program to access the data.









The digital I/O and Pwm light control function will be illustrated in this document. Currently, the library and driver supports the below OS:

- Windows 7 (32-bit & 64-bit)
- Windows 8 (32-bit & 64-bit)
- Windows 10 (32-bit & 64-bit)

For the other OS to be supported, please contact Nodka support for the further information.

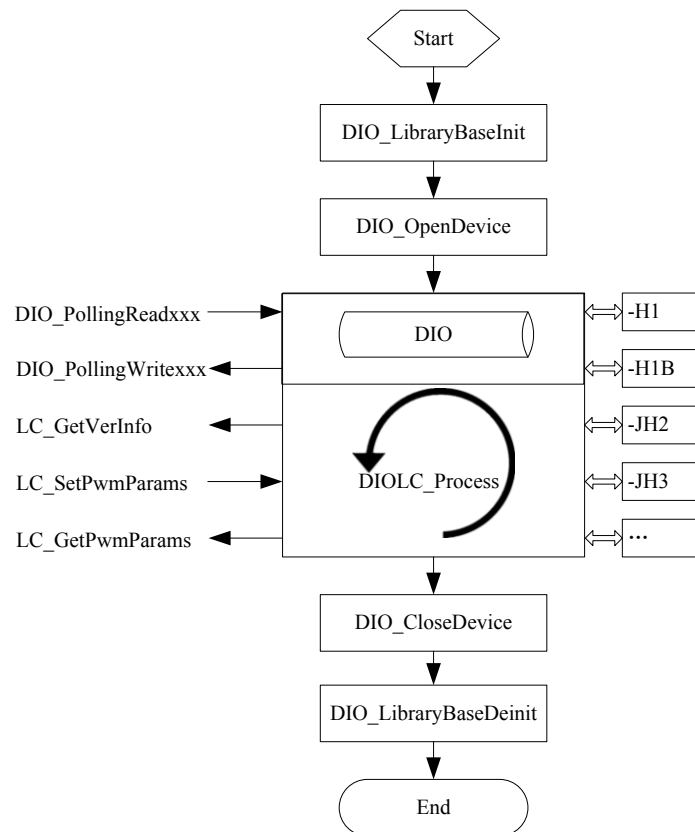
### 3 Installation

The drivers and library files can be decompressed by installing the 'NKDIO\_Driver\_Setup\_x86.exe' file. The default installation path is 'C:\NODKA', a folder named NKDIO\_SDK will be found after the driver installed successfully. The below is the contents list in the NKDIO\_SDK folder.

 Bin	■ Bin: Test tool working folder, which can be used to test the functionality without coding.
 Include	■ Include: the header files including the API declaration, which can be included in the C/C++ development.
 Lib	■ Lib: library binary files which will be linked to the execute files.
 Manual	■ Manual: user manual for the SDK.
 Sample	■ Sample: some examples to be referenced during the development.
 vc_redist	■ vc_redist: VC++ runtime library, you must install them manually if your PC has not installed.
 unins000.dat	
 unins000.exe	

## 4 API usage

The dynamic library can be used to access many add-on boards, the library must be initialized before connecting to the device and accessing the data, and library process function must be called cyclically to execute the command and communicate with the devices. There is a profile file which stored the configure information for the board, which must be imported during the library initialization. The blow is the logic schematic for your reference.



### 4.1 API function list

Function Name	Description
<b>Library Base</b>	
DIOLC_LibraryBaseInit	Init the library and the drivers
DIOLC_OpenDevice	Open the device port, for the light control, the port is serial port 3
DIOLC_CloseDevice	Close and disconnect to the port
DIOLC_IsDeviceOpened	Check if the port is opened or not
DIOLC_Process	Notify function to process the command
DIOLC_LibraryBaseDeinit	Deinitialize the library and release the resources
DIOLC_SetGeneralParam	Set the general parameter for the controller
DIOLC_GetGeneralParam	Get the general parameter from the controller.
<b>DIO in polling mode</b>	
DIO_PollingReadDiBit	Read the DI channel in the polling mode
DIO_PollingReadDIByte	Read the DI port in the polling mode
DIO_PollingWriteDoBit	Write the DO channel in the polling mode
DIO_PollingWriteDoByte	Write the DO port in the polling mode
DIO_PollingReadDoBit	Read back the set value of the digital output channel in the polling mode.
DIO_PollingReadDoByte	Read back the set value of the digital output port(8 bits).
<b>PWM Light Control</b>	
LC_GetVerInfo	Read the version information of the hardware and firmware
LC_SetPwmParamsNoRetentive	Send the parameters to turn on/off the specified light channel

	without saving the parameters to the controller.
LC_SetPwmParams	Send the parameters to turn on/off the specified light channel and the parameters will be saved to the controller.
LC_GetPwmParams	Read the parameters of the specified light channel from the controller

## 4.2 Library Base

### 4.2.1 DIOLC\_LibraryBaselnit

- **Description**  
Library initialized.
- **Prototype**  
`int DIOLC_LibraryBaselnit(const char * configFile)`
- **Parmeters**
  - Input
    - ◆ `configIniFile`: the name and path of the configure file. The configuration for the IO or light control for each add-on board is stored in the ini format file, which can be got in the according folder after the SDK is installed.
  - Output
    - ◆ None
- **Return**  
Return 1 if success; return -1 when failed.
- **Other**  
The function should be called firstly but only once to allocate the resources before using the I/O access functions in the library.
- **Usage**  
`int ret = DIOLC_LibraryBaselnit (".\\nkio_config.ini");`

### 4.2.2 DIOLC\_OpenDevice

- **Description**  
Send the command to open the port.
- **Prototype**  
`int DIOLC_OpenDevice(unsigned short port, pLcCallbackFunc pCallBackFun)`
- **Parmeters**
  - Input
    - ◆ `port`: the port to be opened, for the NP-61xx series products, COM3 is used to communicate with the light controller, so the port should be set to 3 when using the light controller.
    - ◆ `pCallBackFun`: the pointer of the callback function, the function will be called once after the command is executed.
  - Output
    - ◆ None
- **Return**  
Return -1 if failed, and return the other value if success.
- **Other**  
The structure of parameters in the callback function as the below shows. The version of the hardware and firmware will be returned in the parameters when the device is opened successfully. Please remind that you can not execute the block code in the callback functions.  

```
typedef struct _OPENCOM_CALLBACK_ARG_T
{
    unsigned short portNum;           // port number has been opened.
    unsigned char devId;              // the device Id opened
    unsigned char hardwareMajorVer;   // hardware major version returned
    unsigned char hardwareMinorVer;   // hardware minor version returned
    unsigned char hardwareRevVer;     // hardware release version returned
}
```



```

        unsigned char firmwareMajorVer; // firmware major version returned
        unsigned char firmwareMinorVer; // firmware minor version returned
        unsigned char firmwareRevVer;   // firmware release version returned
        unsigned char fillup_1;
        unsigned char fillup_2;
        unsigned char error;            // the error flag, 0:no error, and 1 when open error.
        unsigned int errorId;           // the error ID.
    }OPENCOM_CALLBACK_ARG_T;

```

## ■ Usage

// Open port callback function

```
void LcOpenPortCallback(LC_CALLBACK_ARG_T arg)
```

```

{
    If (arg.openComCallbackArg.error)
    {
        printf("Open port %d failed: %d\n\r",
               arg.openComCallbackArg.portNum,
               arg.openComCallbackArg.errorId);
    }
    else
    {
        printf("Hardware version: %d.%d.%d Firmware version: %d.%d.%d\n\r",
               arg.openComCallbackArg.hardwareMajorVer,
               arg.openComCallbackArg.hardwareMinorVer,
               arg.openComCallbackArg.hardwareRevVer,
               arg.openComCallbackArg.firmwareMajorVer,
               arg.openComCallbackArg.firmwareMinorVer,
               arg.openComCallbackArg.firmwareRevVer);
    }
}

```

// Server thread

```
DWORD WINAPI ServerThread(LPVOID lpParameter)
```

```

{
    While(1)
    {
        DIOLC_Process();
        Sleep(1);
    }
    return 0;
}

```

```
HANDLE xServerThreadHdl = NULL;
```

```
DWORD dwServerThreadId;
```

```
int main()
```

```

{
    int ret = DIOLC_LibraryBaseInit("./nkio_config.ini");
    if( ret >= 0)
    {
        xServerThreadHdl = CreateThread(NULL,1024,ServerThread,NULL,0,& dwServerThreadId);
        if(NULL == xServerThreadHdl)
        {
            printf("Create server thread failed\n\r");
        }
        else
        {
            // open the port 3
            DIOLC_OpenDevice (3, LcOpenPortCallback);
        }
        .....
    }
}

```

```

    }
    return 0;
}

```

#### 4.2.3 DIOLC\_CloseDevice

##### ■ Description

Send the command to close the port.

##### ■ Prototype

```
int DIOLC_CloseDevice(unsigned short port, pLcCallbackFunc pCallbackFun)
```

##### ■ Parameters

###### ➤ Input

- ◆ port: the port to be closed, for the NP-61xx series products, COM3 is used to communicate with the light controller, so the port should be set to 3.
- ◆ pCallbackFun: the pointer of the callback function, the function will be called once after the command is executed.

###### ➤ Output

- ◆ None

##### ■ Return

Return -1 if failed, and return the other value if success.

##### ■ Other

The structure output in the callback function as the below, please remind that you can not execute the block code in the callback functions.

```

typedef struct _CLOSECOM_CALLBACK_ARG_T
{
    unsigned int fill_up;
    unsigned char fill_up_1;
    unsigned char fill_up_2;
    unsigned char fill_up_3;
    unsigned char error;        // error flag, 0: no error, 1: error happened
    unsigned int errorId;      // the error ID if the error happened.
    unsigned int fill_up_4;
}CLOSECOM_CALLBACK_ARG_T;

```

##### ■ Usage

```

// Open port callback function
void LcOpenPortCallback(LC_CALLBACK_ARG_T arg)
{
    If (arg.openComCallbackArg.error)
    {
        Printf("Open port %d failed: %d\n\r",
            arg.openComCallbackArg.portNum,
            arg.openComCallbackArg.errorId);
    }
}

void LcClosePortCallback(LC_CALLBACK_ARG_T arg)
{
    If (arg.closeComCallbackArg.error)
    {
        Printf("Close port failed: %d\n\r",
            arg.closeComCallbackArg.errorId);
    }
}

// Server thread
DWORD WINAPI ServerThread(LPVOID lpParameter)
{

```

```

While(1)
{
    DIOLC_Process();
    Sleep(1);
}
return 0;
}

HANDLE xServerThreadHdl = NULL;
DWORD dwServerThreadId;
int main()
{
    int ret = LC_Init(0);
    if( ret >= 0)
    {
        xServerThreadHdl = CreateThread(NULL,1024,ServerThread,NULL,0,& dwServerThreadId);
        if(NULL == xServerThreadHdl)
        {
            printf("Create server thread failed\n\r");
        }
        else
        {
            // open the port 3
            DIOLC_OpenDevice( 3, LcOpenPortCallback);

            .....

            // close the port if disconnect and exit the application
            DIOLC_CloseDevice(3, LcClosePortCallback);
        }
        .....
    }
    return 0;
}

```

#### 4.2.4 DIOLC\_IsDeviceOpened

- **Description**  
Check if the port is opened.
- **Prototype**  
int DIOLC\_IsDeviceOpened(unsigned short port, unsigned char devId)
- **Parameters**
  - Input
    - ◆ port: the port number to be checked.
    - ◆ devId: the Id of the device, set to 0x01 for the light controller.
  - Output
    - ◆ None
- **Return**  
Return 0 if not opened, otherwise return 1.
- **Other**  
Only used to check the serial port.
- **Usage**  
int ret = DIOLC\_IsDeviceOpened(3, 0x01); // check whither the COM3 is opened or not.

#### 4.2.5 DIOLC\_Process

- **Description**  
The notification to execute the command in the stack and communicate with the device by the serial port. This function must be called cyclically in the server thread.

- **Prototype**  
int DIOLC\_Process(void)
- **Parameters**
  - Input
    - ◆ None
  - Output
    - ◆ None
- **Return**  
Return -1 if error, other value when success.
- **Other**  
The function must be called cyclically in the server thread.
- **Usage**  

```
int ret = DIOLC_LibraryBaseInit("nkio_config.ini");
if( ret >= 0)
.....
while(1)
{
    DIOLC_Process();
    Sleep(1);
}
```

#### 4.2.6 DIOLC\_LibraryBaseDeinit

- **Description**  
The function to be used to release the library resources before the application exit.
- **Prototype**  
int DIOLC\_LibraryBaseDeinit(void)
- **Parameters**
  - Input
    - ◆ None
  - Output
    - ◆ None
- **Return**  
Return -1 if error, other value when success.
- **Other**  
The function should be called before the application exit.
- **Usage**  

```
int ret = DIOLC_LibraryBaseInit("nkio_config.ini");
if( ret >= 0)
.....
while(1)
{
    DIOLC_Process();
    Sleep(1);
}

.....
DIOLC_LibraryBaseDeinit();
```

#### 4.2.7 DIOLC\_SetGeneralParam

- **Description**  
Set general parameters to the controller, please refer to the parameter table supported.
- **Prototype**  

```
int DIOLC_SetGeneralParam(unsigned int devId,
    unsigned char ucParamId,
    unsigned char ucParamLen,
    unsigned int uiParamValue,
```

pLcCallbackFunc pCallbackFun)

## ■ Parameters

### ➤ Input

- ◆ devId: the Id of the device, set to 0x01 for the light controller.
- ◆ ucParamId: the ID of the parameter to be set.
- ◆ ucParamLen: the length of the parameter to be set.
- ◆ uiParamValue: the value of the parameter to be set.
- ◆ pCallbackFun: the pointer of the callback function, the function will be called once after the command is executed.

### ➤ Output

- ◆ None

## ■ Return

Return -1 if failed, and return the other value if success.

## ■ Other

The structure of parameters in the callback function as the below shows. The parameter information will be returned in the call back function once the parameter is set successfully. Please remind that you can not execute the block code in the callback functions.

```
typedef struct _SET_GENERAL_PARAM_CALLBACK_ARG_T
{
    unsigned char devId;           // The ID of the device, 0x01 is for the NP add-on IO board.
    unsigned char cmdId;          // Ignore.
    unsigned char paramId;        // The parameter ID should be set in the controller.
    unsigned char paramLen;       // The length of the parameter value in the controller.
    unsigned int paramValue;       // The value of the parameter has been set.
    unsigned char fill_up_1;
    unsigned char fill_up_2;
    unsigned char fill_up_3;
    unsigned char error;          // Error flag when set failed, 0 is successful.
    unsigned int errorId;         // The error ID when the error happened.
}SET_GENERAL_PARAM_CALLBACK_ARG_T;
```

## ■ Parameter table

ID	Parameter	Value	R/W	Length(Byte)	Description
1	Device ID	1~255	R	1 Byte	For the light control IO board, it is 1.
2	Hardware version Major	0~255	R	1 Byte	
3	Hardware version Minor	0~255	R	1 Byte	
4	Hardware version Release	0~255	R	1 Byte	
5	Firmware version Major	0~255	R	1 Byte	
6	Firmware version Minor	0~255	R	1 Byte	
7	Firmware version Release	0~255	R	1 Byte	
10	Light Control Channel 0 holding time unit	0: 1000ms 1: 100ms 2: 10ms 3: 1ms	R/W	1 Byte	Only valid in the edge triggering mode.
11	Light Control Channel 1 holding time unit	0: 1000ms 1: 100ms 2: 10ms 3: 1ms	R/W	1 Byte	Only valid in the edge triggering mode.
12	Light Control Channel 2 holding time unit	0: 1000ms 1: 100ms	R/W	1 Byte	Only valid in the edge triggering mode.

		2: 10ms 3: 1ms			
13	Light Control Channel 3 holding time unit	0: 1000ms 1: 100ms 2: 10ms 3: 1ms	R/W	1 Byte	Only valid in the edge triggering mode.

Table 1 General parameter

#### ■ Usage

// Set the general parameter callback.

```
void SetGeneralParameterCallback(LC_CALLBACK_ARG_T arg)
{
    If (arg.setGeneralParamCallbackArg.error)
    {
        printf("Set general parameter %d to %d failed\n\r",
            arg.setGeneralParamCallbackArg.paramId,
            arg.setGeneralParamCallbackArg.paramValue);
    }
    else
    {
        printf("Set general parameter %d to %d success\n\r",
            arg.setGeneralParamCallbackArg.paramId,
            arg.setGeneralParamCallbackArg.paramValue);
    }
}
```

// Open port callback function

```
void LcOpenPortCallback(LC_CALLBACK_ARG_T arg)
{
    If (arg.openComCallbackArg.error)
    {
        printf("Open port %d failed: %d\n\r",
            arg.openComCallbackArg.portNum,
            arg.openComCallbackArg.errorId);
    }
    else
    {
        printf("Hardware version: %d.%d.%d Firmware version: %d.%d.%d\n\r",
            arg.openComCallbackArg.hardwareMajorVer,
            arg.openComCallbackArg.hardwareMinorVer,
            arg.openComCallbackArg.hardwareRevVer,
            arg.openComCallbackArg.firmwareMajorVer,
            arg.openComCallbackArg.firmwareMinorVer,
            arg.openComCallbackArg.firmwareRevVer);
    }
}
```

// Server thread

```
DWORD WINAPI ServerThread(LPVOID lpParameter)
{
    While(1)
    {
        DIOLC_Process();
        Sleep(1);
    }
    return 0;
}
```

```

HANDLE xServerThreadHdl = NULL;
DWORD dwServerThreadId;
int main()
{
    int ret = DIOLC_LibraryBaseInit("./nkio_config.ini");
    if( ret >= 0)
    {
        xServerThreadHdl = CreateThread(NULL, 1024, ServerThread, NULL, 0, & dwServerThreadId);
        if(NULL == xServerThreadHdl)
        {
            printf("Create server thread failed\n\r");
        }
        else
        {
            // open the port 3
            DIOLC_OpenDevice (3, LcOpenPortCallback);

            .....
            // Set the holding time unit of the channel 0 to 100ms
            DIOLC_SetGeneralParam(0x01, // DevId
                                10, // Parameter ID
                                1, // Parameter Length
                                1, // Parameter Value
                                SetGeneralParameterCallback // Operate call back
                                );

            .....

        }
        .....
    }
    return 0;
}

```

#### 4.2.8 DIOLC\_GetGeneralParam

##### ■ Description

Get general parameters from the controller, please refer to the parameter table supported.

##### ■ Prototype

```

int DIOLC_GetGeneralParam(unsigned int devId,
                          unsigned char ucParamId,
                          unsigned char ucParamLen,
                          pLcCallbackFunc pCallBackFun)

```

##### ■ Parameters

###### ➤ Input

- ◆ devId: the Id of the device, set to 0x01 for the light controller.
- ◆ ucParamId: the ID of the parameter to be read.
- ◆ ucParamLen: the length of the parameter to be read.
- ◆ pCallBackFun: the pointer of the callback function, the function will be called once after the command is executed.

###### ➤ Output

- ◆ None

##### ■ Return

Return -1 if failed, and return the other value if success.

##### ■ Other

The structure of parameters in the callback function as the below shows. The parameter information will be returned in the callback function once the parameter is read successfully. Please remind that you can not execute the block code in the callback functions.

```
typedef struct _GET_GENERAL_PARAM_CALLBACK_ARG_T
{
    unsigned char devId;        // The ID of the device, 0x01 is for the NP add-on IO board.
    unsigned char cmdId;        // Ignore.
    unsigned char paramId;     // The parameter ID should be got from the controller.
    unsigned char paramLen;    // The length of the parameter value.
    unsigned int paramValue;    // The value of the parameter has been got.
    unsigned char fill_up_1;
    unsigned char fill_up_2;
    unsigned char fill_up_3;
    unsigned char error;        // Error flag when set failed, 0 is successful.
    unsigned int errorId;       // The error ID when the error happened.
}GET_GENERAL_PARAM_CALLBACK_ARG_T;
```

#### ■ Parameter table

ID	Parameter	Value	R/W	Length(Byte)	Description
1	Device ID	1~255	R	1 Byte	For the light control IO board, it is 1.
2	Hardware version Major	0~255	R	1 Byte	
3	Hardware version Minor	0~255	R	1 Byte	
4	Hardware version Release	0~255	R	1 Byte	
5	Firmware version Major	0~255	R	1 Byte	
6	Firmware version Minor	0~255	R	1 Byte	
7	Firmware version Release	0~255	R	1 Byte	
10	Light Control Channel 0 holding time unit	0: 1000ms 1: 100ms 2: 10ms 3: 1ms	R/W	1 Byte	Only valid in the edge triggering mode.
11	Light Control Channel 1 holding time unit	0: 1000ms 1: 100ms 2: 10ms 3: 1ms	R/W	1 Byte	Only valid in the edge triggering mode.
12	Light Control Channel 2 holding time unit	0: 1000ms 1: 100ms 2: 10ms 3: 1ms	R/W	1 Byte	Only valid in the edge triggering mode.
13	Light Control Channel 3 holding time unit	0: 1000ms 1: 100ms 2: 10ms 3: 1ms	R/W	1 Byte	Only valid in the edge triggering mode.

Table 2 General parameter

#### ■ Usage

```
// Get the general parameter callback.
void GetGeneralParameterCallback(LC_CALLBACK_ARG_T arg)
{
    If (arg. getGeneralParamCallbackArg.error)
```



```
{
    printf("Get general parameter %d failed\n\r",
        arg. getGeneralParamCallbackArg.paramId);
}
else
{
    printf("Get general parameter %d = %d success\n\r",
        arg. getGeneralParamCallbackArg.paramId,
        arg. getGeneralParamCallbackArg.paramValue);
}
}

// Open port callback function
void LcOpenPortCallback(LC_CALLBACK_ARG_T arg)
{
    If (arg.openComCallbackArg.error)
    {
        printf("Open port %d failed: %d\n\r",
            arg.openComCallbackArg.portNum,
            arg.openComCallbackArg.errorId);
    }
    else
    {
        printf("Hardware version: %d.%d.%d Firmware version: %d.%d.%d\n\r",
            arg.openComCallbackArg.hardwareMajorVer,
            arg.openComCallbackArg.hardwareMinorVer,
            arg.openComCallbackArg.hardwareRevVer,
            arg.openComCallbackArg.firmwareMajorVer,
            arg.openComCallbackArg.firmwareMinorVer,
            arg.openComCallbackArg.firmwareRevVer);
    }
}

// Server thread
DWORD WINAPI ServerThread(LPVOID lpParameter)
{
    While(1)
    {
        DIOLC_Process();
        Sleep(1);
    }
    return 0;
}

HANDLE xServerThreadHdl = NULL;
DWORD dwServerThreadId;
int main()
{
    int ret = DIOLC_LibraryBaseInit("./nkio_config.ini");
    if( ret >= 0)
    {
        xServerThreadHdl = CreateThread(NULL,1024,ServerThread,NULL,0,& dwServerThreadId);
        if(NULL == xServerThreadHdl)
        {
            printf("Create server thread failed\n\r");
        }
        else
        {
            // open the port 3

```

```

DIOLC_OpenDevice (3, LcOpenPortCallback);
.....
// Get the holding time unit of the channel 0
DIOLC_GetGeneralParam(0x01, // DevId
                      10, // Parameter ID
                      1, // Parameter Length
                      GetGeneralParameterCallback // Operate call back
                      );

.....
}
.....
}
return 0;
}

```

## 4.3 DIO function

### 4.3.1 Polling Mode

The library provide the functions to access the digital I/O in the polling mode, but in the case of multi-thread access the independent I/O channel, the critical mutex must be used to protect the resources.

#### 4.3.1.1 DIO\_PollingReadDiBit

- **Description**  
Read a single DI channel in the polling mode.
- **Prototype**  
BOOL DIO\_PollingReadDiBit(BYTE diByteIndex, BYTE diBitIndex)
- **Parameters**
  - Input
    - ◆ diByteIndex: The byte index of the digital input channel located, for the first 8 channel, the diByteIndex is set to 0, while the 8~15 channel is set to 1.
    - ◆ diBitIndex: The bit index in the according byte (start from 0).
  - Output
    - ◆ None
- **Return**  
The status (TRUE or FALSE) of the specified DI channel.
- **Other**  
The critical signal must be used when called in the muti-thread.
- **Usage**

```

if(0 != DIOLC_LibraryBaseInit("./nkio_config.ini"))
{
    BOOL DI3 = DIO_PollingReadDiBit(0, 3); // access the DI3 status.
    BOOL DI9 = DIO_PollingReadDiBit(1, 1); // access the DI9 status.
}

```

#### 4.3.1.2 DIO\_PollingReadDiByte

- **Description**  
Read digital input port value( 8 channels) in the polling mode.
- **Prototype**  
BOOL DIO\_PollingReadDiByte(BYTE diByteIndex)
- **Parameters**
  - Input

- ◆ diByteIndex: The byte index of the digital input channel located, for the first 8 channel, the diByteIndex is set to 0, while the 8~15 channel is set to 1.
- Output
  - ◆ None
- **Return**  
The value of digital input port, (one bit represent for one channel).
- **Other**  
The critical signal must be used when called in the muti-thread.
- **Usage**  

```

BYTE port0 = 0;
BYTE port1 = 1;
if(0 != DIOLC_LibraryBaseInit("./nkio_config.ini"))
{
    BYTE port0Value = DIO_PollingReadDiByte(port0); // access the status of the first 8 channel .
    BYTE port1Value = DIO_PollingReadDiByte(port1); // access the status of the second 8
channels if supported.
}
      
```

#### 4.3.1.3 DIO\_PollingWriteDoBit

- **Description**  
Set the status of the speacial digital output channel in the polling mode.
- **Prototype**  
VOID DIO\_PollingWriteDiBit(BYTE doByteIndex, BYTE doBitIndex, BYTE doBitValue)
- **Parameters**
  - Input
    - ◆ doByteIndex: The byte index of the digital output channel located, for the first 8 channel, the doByteIndex is set to 0, while the 8~15 channel is set to 1.
    - ◆ doBitIndex: The bit index in the according byte (start from 0).
    - ◆ doBitValue: the status to be set, set to 1 to reset and set to 0 when set the channel.
  - Output
    - ◆ None
- **Return**  
None.
- **Other**  
The critical signal must be used when called in the muti-thread.
- **Usage**  

```

BYTE chByteIdx = 0;
BYTE chBitIdx = 1;
BYTE chValue = 1;
if(0 != DIOLC_LibraryBaseInit("./nkio_config.ini"))
{
    DIO_PollingWriteDoBit(chByteIdx, chBitIdx, chValue); // set the status of DO1.
    DIO_PollingWriteDoBit(0, 2, 0); // ReSet the status of DO2.
}
      
```

#### 4.3.1.4 DIO\_PollingWriteDoByte

- **Description**  
Write the output port value (8 channels).
- **Prototype**  
VOID DIO\_PollingWriteDiByte(BYTE doByteIndex, BYTE doByteValue)
- **Parameters**
  - Input
    - ◆ doByteIndex: The byte index of the digital output channel located, for the first 8 channel, the doByteIndex is set to 0, while the 8~15 channel is set to 1.
    - ◆ doByteValue: the port status to be set, from 0 to 0xFF. 0x0 to turn on all of the port, and set to 0xFF to reset the port.

- Output
  - ◆ None
- **Return**  
None.
- **Other**  
The critical signal must be used when called in the muti-thread.
- **Usage**  

```

BYTE portIdx = 0;
BYTE portValue = 0x0F;
if(0 != DIOLC_LibraryBaseInit("./nkio_config.ini"))
{
    DIO_PollingWriteDoByte(portIdx, portValue); // set the value of port 0.
}
      
```

#### 4.3.1.5 DIO\_PollingReadDoBit

- **Description**  
Read back the digital output bit value has been set.
- **Prototype**  

```

BOOL DIO_PollingReadDoBit(BYTE doByteIndex, BYTE doBitIndex)
      
```
- **Parmeters**
  - Input
    - ◆ doByteIndex: The byte index of the digital output channel located, for the first 8 channel, the doByteIndex is set to 0, while the 8~15 channel is set to 1.
    - ◆ doBitIndex: The bit index in the according byte (start from 0 to 7).
  - Output
    - ◆ None
- **Return**  
The digital output channel status.
- **Other**  
The critical signal must be used when called in the muti-thread.
- **Usage**  

```

BYTE portIdx = 0;
BYTE bitIdx = 5;
if(0 != DIOLC_LibraryBaseInit("./nkio_config.ini"))
{
    BOOL status = DIO_PollingReadDoBit(portIdx, bitIdx); // Read the status of the channel 5.
}
      
```

#### 4.3.1.6 DIO\_PollingReadDoByte

- **Description**  
Read back the digital output port value has been set.
- **Prototype**  

```

BYTE DIO_PollingReadDoByte(BYTE doByteIndex)
      
```
- **Parmeters**
  - Input
    - ◆ doByteIndex: The byte index of the digital output channel located, for the first 8 channel, the doByteIndex is set to 0, while the 8~15 channel is set to 1.
  - Output
    - ◆ None
- **Return**  
The digital output port status, one port include 8 channels.
- **Other**  
The critical signal must be used when called in the muti-thread.
- **Usage**  

```

BYTE portIdx = 0;
if(0 != DIOLC_LibraryBaseInit("./nkio_config.ini"))
{
    BYTE doPortValue = DIO_PollingReadDoByte(portIdx); // Read the status of the channel 5.
}
      
```

## 4.4 Light Control function

Since asynchronous communication mechanism is used in the serial communication, so all of the command will be pushed in the command stack, and a higher priority thread must be created to calling the notification process function to execute the command in the stack and return the result using the callback functions, the user application don't need to wait the result after the command send out. Please remind that the block code must not executed in the callback functions.

### 4.4.1.1 LC\_GetVerInfo

#### ■ Description

The function is used to get the version information of the hardware and the firmware.

#### ■ Prototype

```
int LC_GetVerInfo(unsigned int devId, pLcCallbackFunc pCallBackFun)
```

#### ■ Parameters

##### ➤ Input

- ◆ devId: the Id of the device, set to 0x01 for the light controller.
- ◆ pCallBackFun: the pointer of the callback function, the function will be called once after the command is executed.

##### ➤ Output

- ◆ None

#### ■ Return

Return -1 if failed, and return the value larger than zero if success.

#### ■ Other

It is very important to check the version of the hardware and the firmware when the functionality is not correct. The version information is output by the callback function, the structure as the below:

```
typedef struct _GET_DEVICE_VER_CALLBACK_ARG_T
{
    unsigned char hardwareMajorVer; // hardware major version
    unsigned char hardwareMinorVer; // hardware minor version
    unsigned char hardwareRevVer; // hardware reversion
    unsigned char firmwareMajorVer; // firmware major version
    unsigned char firmwareMinorVer; // firmware minor version
    unsigned char firmwareRevVer; // firmware reversion
    unsigned char fill_up;
    unsigned char error; // error flag, 0: no error, 1: error happened
    unsigned int errorId; // the error ID when the error happened
    unsigned int fill_up_2;
}GET_DEVICE_VER_CALLBACK_ARG_T;
```

#### ■ Usage

```
// Open port callback function
void LcOpenPortCallback(LC_CALLBACK_ARG_T arg)
{
    If (arg.openComCallbackArg.error)
    {
        Printf("Open port %d failed: %d\n\r",
            arg. openComCallbackArg.portNum,
            arg. openComCallbackArg.errorId);
    }
}

void LcClosePortCallback(LC_CALLBACK_ARG_T arg)
{
    If (arg.closeComCallbackArg.error)
    {
        Printf("Close port failed: %d\n\r",
            arg. closeComCallbackArg.errorId);
    }
}
```

```

}

void LcGetVerInfoCallback(LC_CALLBACK_ARG_T arg)
{
    If(arg.getDeviceVerCallbackArg.error)
    {
        printf("Get device version failed: %d\n\r", arg->getDeviceVerCallbackArg.errorId);
    }
    else
    {
        printf("HardwareVersion: %d.%d.%d\n\r",
               arg.getDeviceVerCallbackArg.hardwareMajorVer,
               arg.getDeviceVerCallbackArg.hardwareMinorVer,
               arg.getDeviceVerCallbackArg.hardwareRevVer);
        printf("FirmwareVersion: %d.%d.%d\n\r",
               arg.getDeviceVerCallbackArg.firmwareMajorVer,
               arg.getDeviceVerCallbackArg.firmwareMinorVer,
               arg.getDeviceVerCallbackArg.firmwareRevVer);
    }
}

// Server thread
DWORD WINAPI ServerThread(LPVOID lpParameter)
{
    While(1)
    {
        DIOLC_Process();
        Sleep(1);
    }
    return 0;
}

HANDLE xServerThreadHdl = NULL;
DWORD dwServerThreadId;
int main()
{
    int ret = DIOLC_libraryBaseInit("./nkio_config.ini");
    if( ret >= 0)
    {
        xServerThreadHdl = CreateThread(NULL,1024,ServerThread,NULL,0,& dwServerThreadId);
        if(NULL == xServerThreadHdl)
        {
            printf("Create server thread failed\n\r");
        }
        else
        {
            // open the port 3
            DIOLC_OpenDevice( 3, LcOpenPortCallback);

            // send the command to get the version information
            LC_GetVerInfo(0x01, LcGetVerInfoCallback);
            .....

            // close the port if disconnect and exit the application
            LC_CloseDevice( 3, LcClosePortCallback);
        }
        .....
    }
    return 0;
}

```

#### 4.4.1.2 LC\_SetPwmParamsNoRetentive

##### ■ Description

The function is used to turn on/off the specified light control channel without saving the parameters. **Please note that the parameters will not be lost after reboot when using this API**, this is the main difference between the LC\_SetPwmParams.

##### ■ Prototype

```
int LC_SetPwmParamsNoRetentive(unsigned int devId,
    unsigned char ucChIdx,
    unsigned char ucPwmMode,
    unsigned char ucPwmValue,
    unsigned char ucPwmHoldingTime,
    unsigned char ucPwmOnOff,
    pLcCallbackFunc pCallbackFun)
```

##### ■ Parameters

###### ➤ Input

- ◆ devId: the Id of the device, set to 0x01 for the light controller.
- ◆ ucChIdx: the channel of the light to be set.
  - CH0: 0x01
  - CH1: 0x02
  - CH2: 0x04
  - CH3: 0x08).
- ◆ ucPwmMode: the light trigger mode.
  - 0: Soft switch, Control the light by the PwmOnOff value.
  - 1: Hard switch, triggered by the external sensor, the light is on when the trigger input is on, the light is off while the trigger input is off.
  - 2: External trigger on but delay off. The light is triggered when the trigger input on the rising edge, but the duration is set by the holdingtime.
  - 3: Soft Trigger, the light is on when turning on the light by the API, and will keep on with the holding time set, and then will be off automatically.
- ◆ ucPwmValue: the brightness level of the light, from 0 to 100.
- ◆ ucPwmHoldingTime: the duration time when the light working in the mode raising edge mode(2), it is meaningless in the other mode. Unit: second is default, but can be changed by the general parameters.
- ◆ ucPwmOnOff: the switch of the light when working in the soft-trigger mode, set to 0 to turn off the light and set 1 to turn on the light, it is meaningless in the other trigger mode.
- ◆ pCallbackFun: the pointer of the callback function, the function will be called once after the command is executed.

###### ➤ Output

- ◆ None

##### ■ Return

Return -1 if failed, and return the value larger than zero if success.

##### ■ Other

The result will be output by the callback function, the structure as the below:

```
typedef struct _SET_PWM_PARAMS_CALLBACK_ARG_T
{
    unsigned int fill_up;
    unsigned char chIdx;        // the channel of the light has been set.
    unsigned char fill_up_1;
    unsigned char fill_up_2;
    unsigned char error;        // error flag, 0: no error, 1: error happened.
    unsigned int errorId;       // the error ID when the error happened.
    unsigned int fill_up_3;
}SET_PWM_PARAMS_CALLBACK_ARG_T;
```

##### ■ Usage

```
// Open port callback function
void LcOpenPortCallback(LC_CALLBACK_ARG_T arg)
```

```

{
    If (arg.openComCallbackArg.error)
    {
        Printf("Open port %d failed: %d\n\r",
            arg.openComCallbackArg.portNum,
            arg.openComCallbackArg.errorId);
    }
}

void LcClosePortCallback(LC_CALLBACK_ARG_T arg)
{
    If (arg.closeComCallbackArg.error)
    {
        Printf("Close port failed: %d\n\r",
            arg.closeComCallbackArg.errorId);
    }
}

void LcSetPwmParamsNoRetentiveCallback(LC_CALLBACK_ARG_T arg)
{
    If(arg.setPwmParamsCallbackArg.error)
    {
        printf("Set Pwm Params failed: %d\n\r", arg.setPwmParamsCallbackArg.errorId);
    }
}

// Server thread
DWORD WINAPI ServerThread(LPVOID lpParameter)
{
    While(1)
    {
        DIOLC_Process();
        Sleep(1);
    }
    return 0;
}

HANDLE xServerThreadHdl = NULL;
DWORD dwServerThreadId;
int main()
{
    int ret = DIOLC_LibraryBaseInit("./nkio_config.ini");
    if( ret >= 0)
    {
        xServerThreadHdl = CreateThread(NULL, 1024, ServerThread, NULL, 0, & dwServerThreadId);
        if(NULL == xServerThreadHdl)
        {
            printf("Create server thread failed\n\r");
        }
        else
        {
            // open the port 3
            DIOLC_OpenDevive(3, LcOpenPortCallback);

            // send the command to set the parameters for the CH0
            LC_SetPwmParamsNoRetentive(0x01,
                0x01,          // set params to CH0
                0x0,          // soft-trigger mode
                80,           // the brightness level is 80%
                5,            // make no sense in the current mode

```



```

1,          // turn on the light using the current parameters
LcSetPwmParamsNoRetentiveCallback);

.....

// close the port if disconnect and exit the application
DIOLC_CloseDevice(3, LcClosePortCallback);

}
.....
}
return 0;
}

```

#### 4.4.1.3 LC\_SetPwmParams

##### ■ Description

The function is used to set the parameters to the specified light control channel. **And the parameters will be saved to the controller.**

##### ■ Prototype

```

int LC_SetPwmParams(unsigned int devId,
    unsigned char ucChIdx,
    unsigned char ucPwmMode,
    unsigned char ucPwmValue,
    unsigned char ucPwmHoldingTime,
    unsigned char ucPwmOnOff,
    pLcCallbackFunc pCallBackFun)

```

##### ■ Parameters

###### ➤ Input

- ◆ devId: the Id of the device, set to 0x01 for the light controller.
- ◆ ucChIdx: the channel of the light to be set.
  - CH0: 0x01
  - CH1: 0x02
  - CH2: 0x04
  - CH3: 0x08).
- ◆ ucPwmMode: the light trigger mode.
  - 0: Soft switch, Control the light by the PwmOnOff value.
  - 1: Hard switch, triggered by the external sensor, the light is on when the trigger input is on, the light is off while the trigger input is off.
  - 2: External trigger on but delay off. The light is triggered when the trigger input on the rising edge, but the duration is set by the holdingtime.
  - 3: Soft Trigger, the light is on when turning on the light by the API, and will keep on with the holding time set, and then will be off automatically.
- ◆ ucPwmValue: the brightness level of the light, from 0 to 100.
- ◆ ucPwmHoldingTime: the duration time when the light working in the mode raising edge mode(2), it is meaningless in the other mode. Unit: second is default, but can be changed by the general parameters.
- ◆ ucPwmOnOff: the switch of the light when working in the soft-trigger mode, set to 0 to turn off the light and set 1 to turn on the light, it is meaningless in the other trigger mode.
- ◆ pCallBackFun: the pointer of the callback function, the function will be called once after the command is executed.

###### ➤ Output

- ◆ None

##### ■ Return

Return -1 if failed, and return the value larger than zero if success.

##### ■ Other

The result will be output by the callback function, the structure as the below:

```

typedef struct _SET_PWM_PARAMS_CALLBACK_ARG_T
{
    unsigned int fill_up;
}

```

```

        unsigned char chIdx;        // the channel of the light has been set.
        unsigned char fill_up_1;
        unsigned char fill_up_2;
        unsigned char error;        // error flag, 0: no error, 1: error happened.
        unsigned int errorId;       // the error ID when the error happened.
        unsigned int fill_up_3;
    }SET_PWM_PARAMS_CALLBACK_ARG_T;

```

#### ■ Usage

// Open port callback function

```
void LcOpenPortCallback(LC_CALLBACK_ARG_T arg)
```

```

{
    If (arg.openComCallbackArg.error)
    {
        Printf("Open port %d failed: %d\n\r",
               arg.openComCallbackArg.portNum,
               arg.openComCallbackArg.errorId);
    }
}

```

```
void LcClosePortCallback(LC_CALLBACK_ARG_T arg)
```

```

{
    If (arg.closeComCallbackArg.error)
    {
        Printf("Close port failed: %d\n\r",
               arg.closeComCallbackArg.errorId);
    }
}

```

```
void LcSetPwmParamsCallback(LC_CALLBACK_ARG_T arg)
```

```

{
    If(arg.setPwmParamsCallbackArg.error)
    {
        printf("Set Pwm Params failed: %d\n\r", arg.setPwmParamsCallbackArg.errorId);
    }
}

```

// Server thread

```
DWORD WINAPI ServerThread(LPVOID lpParameter)
```

```

{
    While(1)
    {
        DIOLC_Process();
        Sleep(1);
    }
    return 0;
}

```

```
HANDLE xServerThreadHdl = NULL;
```

```
DWORD dwServerThreadId;
```

```
int main()
```

```

{
    int ret = DIOLC_LibraryBaseInit("./nkio_config.ini");
    if( ret >= 0)
    {
        xServerThreadHdl = CreateThread(NULL, 1024, ServerThread, NULL, 0, & dwServerThreadId);
        if(NULL == xServerThreadHdl)
        {
            printf("Create server thread failed\n\r");
        }
    }
    else

```

```

{
    // open the port 3
    DIOLC_OpenDevive(3, LcOpenPortCallback);

    // send the command to set the parameters for the CH0
    LC_SetPwmParams(0x01,
        0x01,    // set params to CH0
        0x0,     // soft-trigger mode
        80,      // the brightness level is 80%
        5,       // make no sense in the current mode
        1,       // turn on the light using the current parameters
        LcSetPwmParamsCallback);
    .....

    // close the port if disconnect and exit the application
    DIOLC_CloseDevice(3, LcClosePortCallback);
}
.....
}
return 0;
}

```

#### 4.4.1.4 LC\_GetPwmParams

##### ■ Description

The function is used to get the parameters of the specified light control channel.

##### ■ Prototype

```
int LC_GetPwmParams(unsigned int devId,
    unsigned char ucChIdx,
    pLcCallbackFunc pCallbackFun)
```

##### ■ Parameters

###### ➤ Input

- ◆ devId: the Id of the device, set to 0x01 for the light controller.
- ◆ ucChIdx: the channel of the light to be set.
  - CH0: 0x01
  - CH1: 0x02
  - CH2: 0x04
  - CH3: 0x08).
- ◆ pCallbackFun: the pointer of the callback function, the function will be called once after the command is executed. All of the parameters will be output in the callback function.

###### ➤ Output

- ◆ None

##### ■ Return

Return -1 if failed, and return the value larger than zero if success.

##### ■ Other

The result will be output by the callback function, the structure as the below:

```
typedef struct _GET_PWM_PARAMS_CALLBACK_ARG_T
{
    unsigned char chIdx;    // the specified light channel, CH0: 0x1, CH1: 0x2, CH2: 0x4, CH3: 0x8
    unsigned char pwmMode; // trigger mode, 0:Soft-switch, 1: hardware-switch, 2: Raising-Edge,
    3: Soft-trigger
    unsigned char pwmValue; // Current brightness level
    unsigned char pwmHoldingTime; // the holding time
    unsigned char pwmOnOff;      // the status of the light, 0: off, 1: on
    unsigned char fill_up_2;
    unsigned char fill_up_3;
    unsigned char error;          // error flag, 0: no error, 1: error happened.
    unsigned int errorId;         // error ID when the error happened.
}
```

```
    unsigned int fill_up_4;
}GET_PWM_PARAMS_CALLBACK_ARG_T;
```

#### ■ Usage

// Open port callback function

```
void LcOpenPortCallback(LC_CALLBACK_ARG_T arg)
```

```
{
    If (arg.openComCallbackArg.error)
    {
        Printf("Open port %d failed: %d\n\r",
               arg.openComCallbackArg.portNum,
               arg.openComCallbackArg.errorId);
    }
}
```

```
void LcClosePortCallback(LC_CALLBACK_ARG_T arg)
```

```
{
    If (arg.closeComCallbackArg.error)
    {
        Printf("Close port failed: %d\n\r",
               arg.closeComCallbackArg.errorId);
    }
}
```

```
void LcGetPwmParamsCallback(LC_CALLBACK_ARG_T arg)
```

```
{
    If(arg.getPwmParamsCallbackArg.error)
    {
        printf("get Pwm Params failed: %d\n\r", arg.getPwmParamsCallbackArg.errorId);
    }
    else
    {
        printf("CH%d: Mode:%d, value: %d, HoldingTime: %d, Status: %d\n\r",
               arg.getPwmParamsCallbackArg.chIdx,
               arg.getPwmParamsCallbackArg.pwmMode,
               arg.getPwmParamsCallbackArg.pwmValue,
               arg.getPwmParamsCallbackArg.pwmHoldingTime,
               arg.getPwmParamsCallbackArg.pwmOnOff);
    }
}
```

// Server thread

```
DWORD WINAPI ServerThread(LPVOID lpParameter)
```

```
{
    While(1)
    {
        DIOLC_Process();
        Sleep(1);
    }
    return 0;
}
```

```
HANDLE xServerThreadHdl = NULL;
```

```
DWORD dwServerThreadId;
```

```
int main()
```

```
{
    int ret = DIOLC_LibraryBaseInit("./nkio_config.ini");
    if( ret >= 0)
    {
        xServerThreadHdl = CreateThread(NULL, 1024, ServerThread, NULL, 0, & dwServerThreadId);
        if(NULL == xServerThreadHdl)
    }
}
```

---

```
{
    printf("Create server thread failed\n\r");
}
else
{
    // open the port 3
    DIOLC_OpenDevice( 3, LcOpenPortCallback);

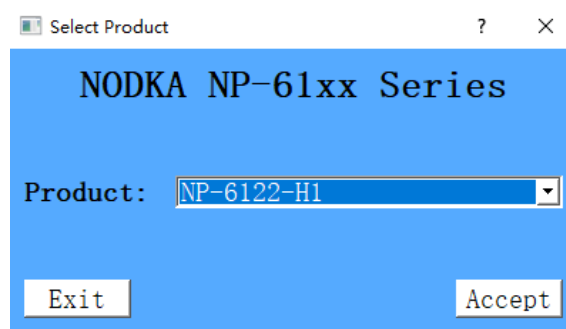
    // send the command to get the parameters of the CH1
    LC_GetPwmParams(0x01,
        0x02,
        LcGetPwmParamsCallback);
    .....

    // close the port if disconnect and exit the application
    DIOLC_CloseDevice( 3, LcClosePortCallback);
}
.....
}
return 0;
}
```

## 5 Test Tool

Target to easily check the IO board firstly before your development, Nodka provide a small tool called NKIO\_TEST\_TOOL.exe which can be used to check the IO working status, it is also can be used to configure some parameters when the IO light controller is working in the external trigger mode. Furthermore, the tool also provides the interface to update the firmware if necessary. The usage of the test tool will be illustrated in this chapter.

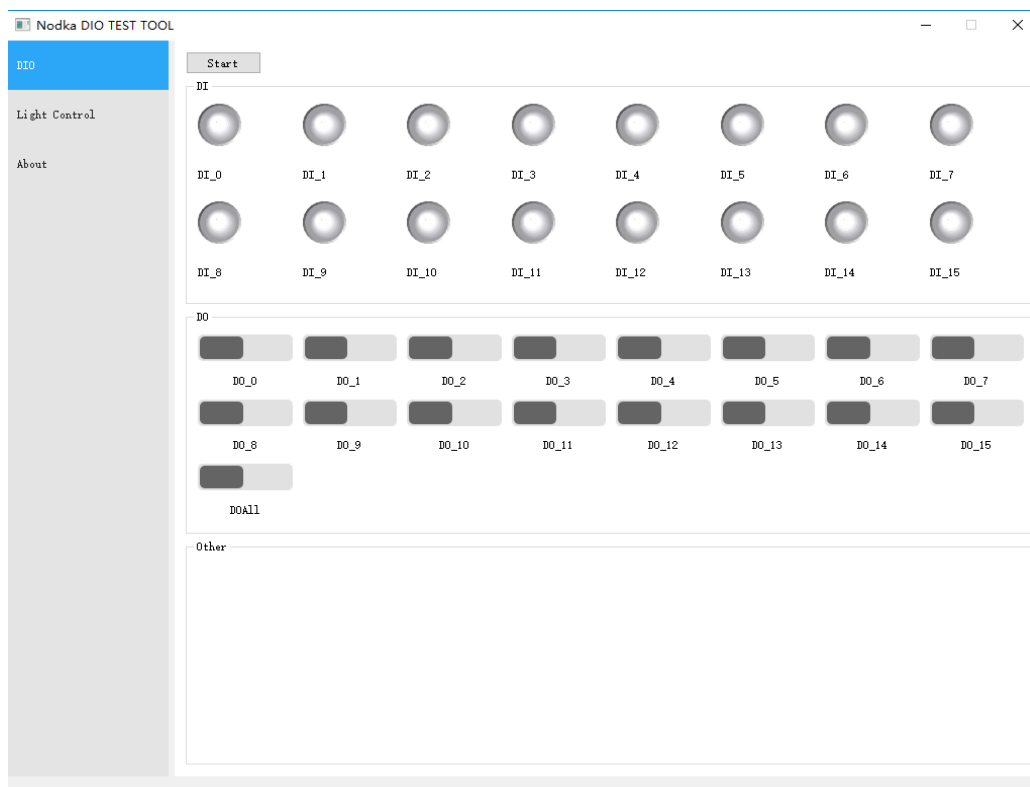
Since the test tool can be used for different I/O addon boards by importing the configuration file, so you must select correct board in the below dialog to import the according profile into the driver.



### 5.1 IO Test

The PC access the digital I/O by writing the value to the according registers directly, and in the I/O test page, one timer is used to refresh the I/O status cyclically after the start button is pushed. The Led will be change to green when the according input is on, and will reset to gray when the digital input is off.

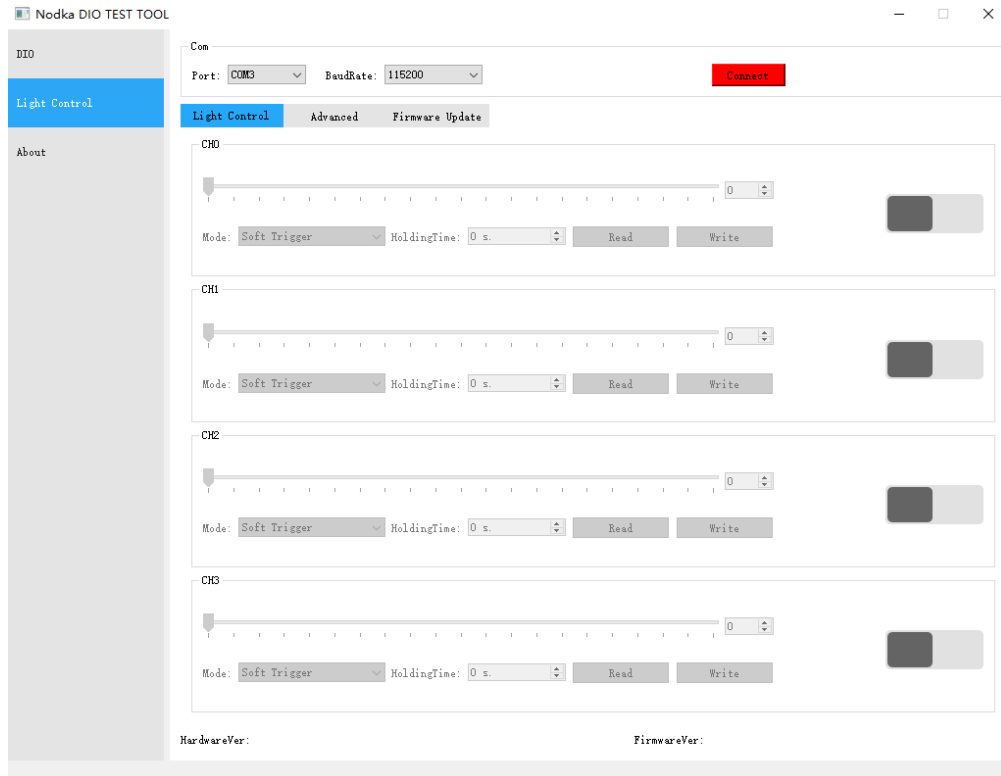
There is one switch button for each digital output channel, you can press the according button to change the digital output status. But for the addon board which only support 8 channels digital output, only the first 8 switch buttons are available, and the others will make no sense when you change the status.



## 5.2 Light Control Test

The PC communicate with the light controller by the serial port 3 in default (Baudrate: 115200, 8 databits, 1 stopbit, no parity) in the current solution.

So the port num is fixed to be COM3 in the current tool, the background color of the connect button will be red if not connected, and will be green when connect successfully. All of the control in the current page are available only in the case of connect successfully.

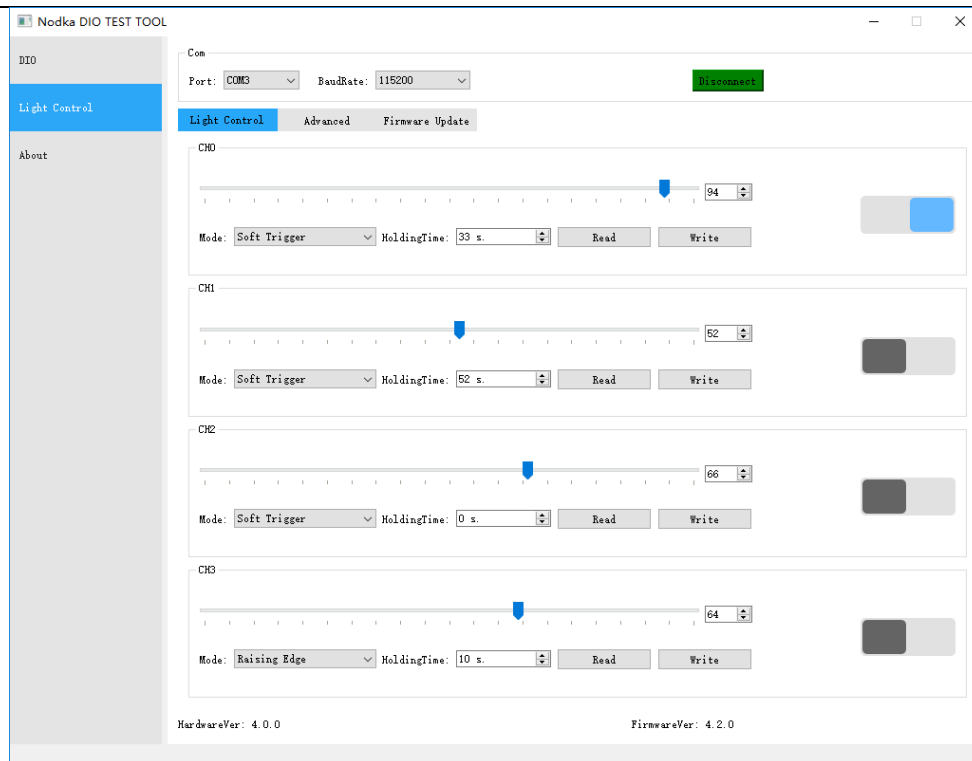


Once the connection is built, the device version information will be displayed, and this tool can only be used on the hardware version 4.0.0 or newer. For the other version, please contact the Nodka support team or reseller to get the detailed information.

In the light control tab, the parameters for each channel will be uploaded in the tool after the connection is built, then parameters for each independent channel can be changed and write into the controller by press the write button or by turning on the switch button on the right in each channel group.

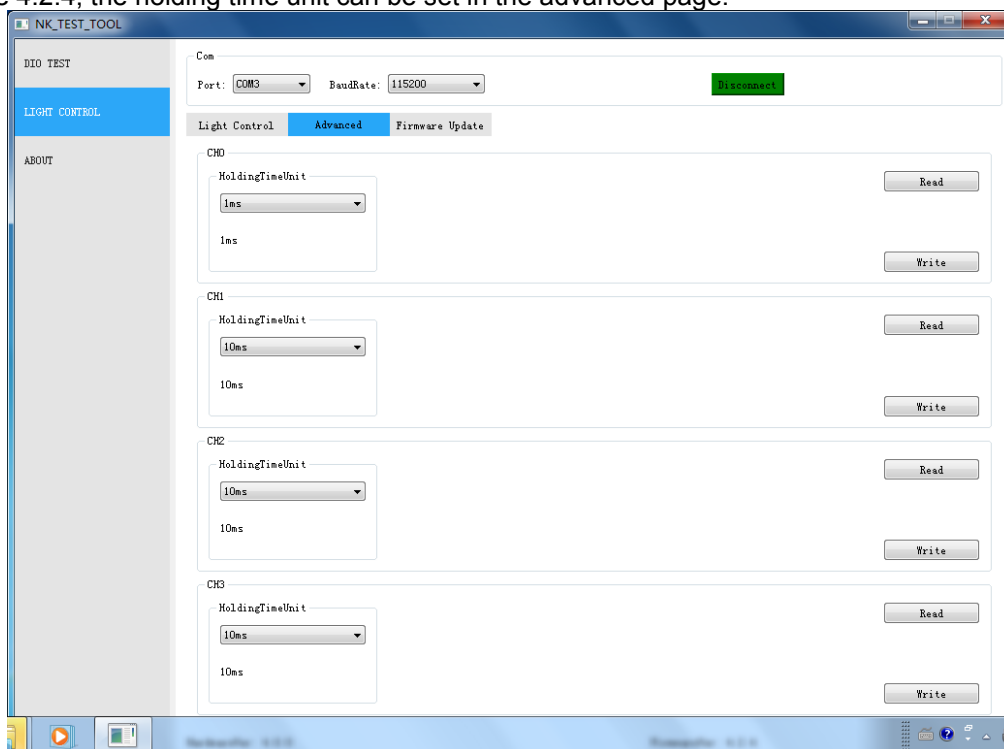
When the lighting controller working in the soft-trigger mode, the switch button can be used to turn on or off the according light channel, and even the brightness level(0~100) can be changed dynamically by the slider bar or spin box when the light is on. The switch button makes no sense in the other hard trigger mode or raising edge mode. The following are the description for the light channels.

- ◆ Mode: the light trigger mode.
  - 0: Soft Switch, set the brightness of the light by the PWM value.
  - 1: Hard Switch, triggered by the external sensor, the light is on when the sensor is on, the light is off while the sensor is off.
  - 2: Raising Edge. The light is triggered when the sensor on the rising edge, but the duration is set by the holdingtime to be set.
- ◆ HoldingTime: the duration time when the light working in the mode raising edge mode(2), it is meaningless in the other mode. Unit: default is second, which can be set from firmware version 4.2.4.



### 5.2.1 Advanced

In the advanced page, you can get and set the advanced parameters for the light controller. From the firmware 4.2.4, the holding time unit can be set in the advanced page.



### 5.3 Firmware update

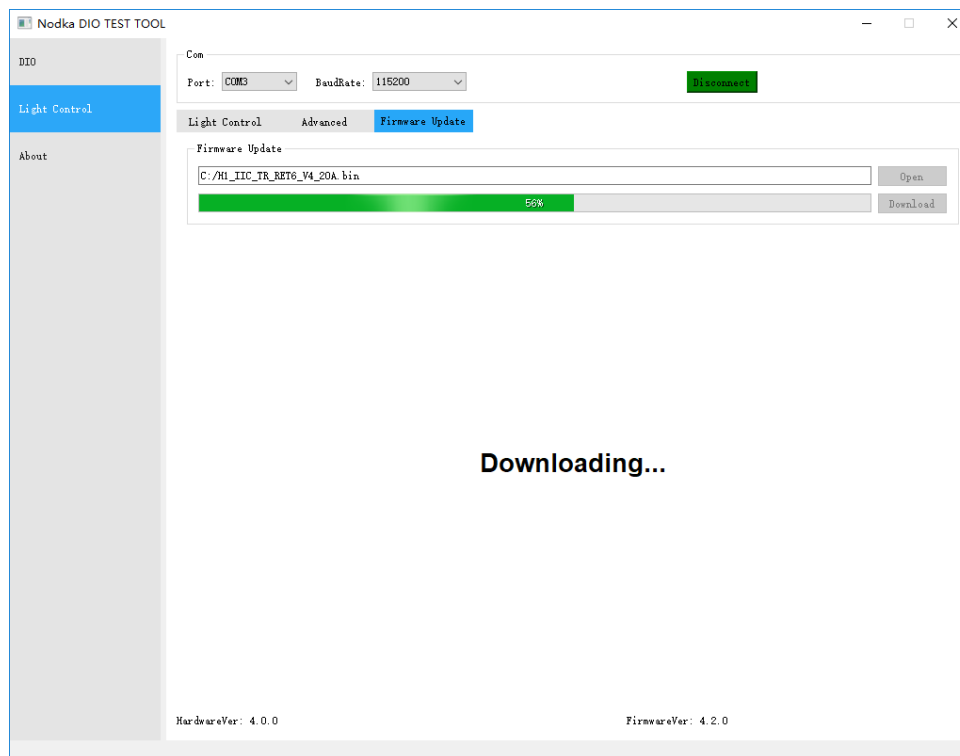
Nodka add-on board also support the IAP functionality to update the firmware of the controller on the line if necessary.

After the serial communication is connected, change to firmware Update tab, you can select the image file on the PC by pressing the open button, then start to download the image file into the controller by





pressing the download button. The progress bar will display the downloading progress, and once finished, the open button will be activated again, and the controller will reboot to use the latest firmware fastly.



Please remind to check the hardware version information and ask for support from Nodka to get the correct firmware image file.

## 6 Development

Nodka provide the dynamic library for the user to develop the programs. You can find the library and the header files in the SDK installation path. Please add the the files in the library folder into the same path with your own application working path.

Please refer to the according sample projects for the detailed information.

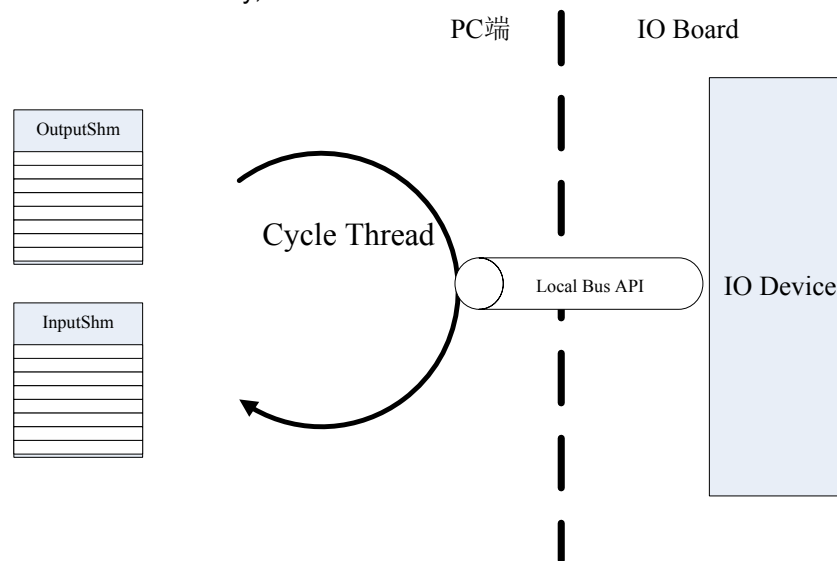
## 7 FAQ

### 7.1 What's the serial communication configure parameters?

On the hardware version of 4.0.0 or new, access the DIO using the PC local bus, but the light control use serial port 3 on the NP series products, the default Baudrate is 115200bps.

### 7.2 How to access the DIO channels in the multi-thread?

Sine the IO devices can only be access by only one thread at a time, so the proposed solution is just create a higher priority server thread as a master to refresh the DI and DO between the shared memory and the IO device, the other threads only need to read the data from the shared memory or write the data to the shared memory, as the below shows.



### 7.3 How to check the hardware version?

You can use the test tool connect to the controller, the version information will be upload and display in the page after connected.

Please remind that the functions in this document only can be used on the hardware version 4.0.0 or newer.

### 7.4 How to get the configure profile for different IO board?

The configuration of the IO board is illustrated in the ini file. In the path of the SDK installation, the path of configure profile is listed in the config.ini file, you must import the according DIO configure profile in the DIO\_Init function during development.

```
config.ini - 记事本
文件(E) 编辑(E) 格式(O) 查看(V) 帮助(H)

# *****
#
#   NKDIO Configuration
#
# *****
[NP-6111-JH2]
ConfigPath = "/NP-6111-JH2/nkio_config.ini"
[NP-6111-JH3]
ConfigPath = "/NP-6111-JH3/nkio_config.ini"
[NP-6122-H1]
ConfigPath = "/NP-6122-H1/nkio_config.ini"
[NP-6122-H1B]
ConfigPath = "/NP-6122-H1B/nkio_config.ini"
[NP-6122-JH2]
ConfigPath = "/NP-6122-JH2/nkio_config.ini"
[NP-6122-JH3]
ConfigPath = "/NP-6122-JH3/nkio_config.ini"

第 1 行, 第 1 列    100%    Windows (CRLF)    UTF-8
```