PHPoC

Device Programming Guide

Version 1.0

Sollae Systems Co., Ltd.

PHPoC Forum: http://www.phpoc.com

Homepage: http://www.sollae.co.kr

Contents

1 Overview

1.1 Device

Physical device and software functions that PHPoC equips are called "device". Every device provides as a special file form and can be used like a general file such as reading/writing files.

Structure of PHPoC file system is as follows:

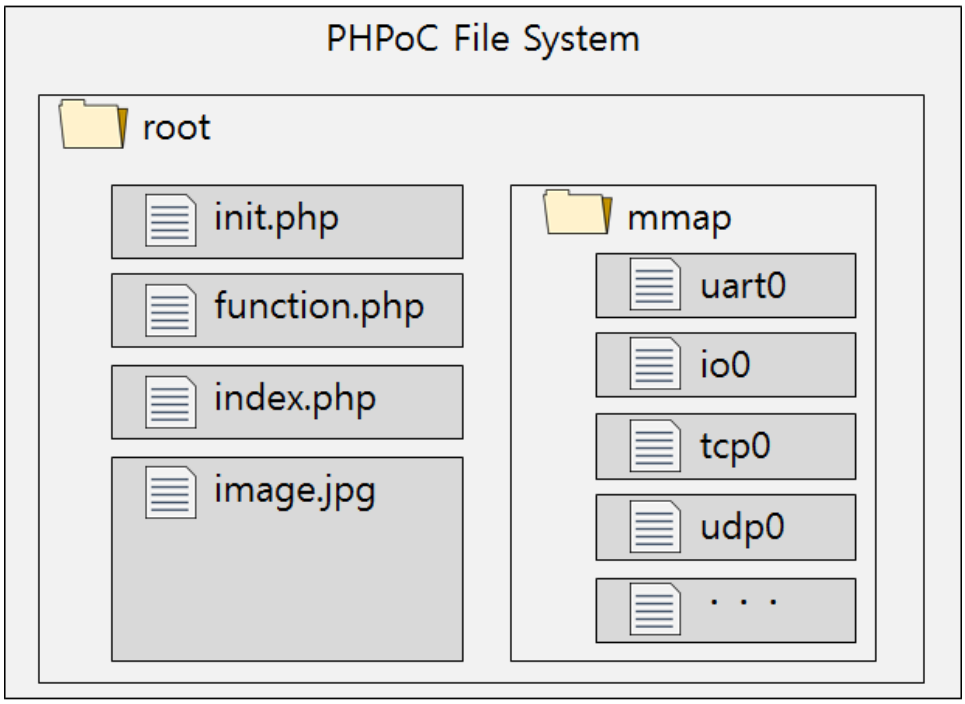


Figure 1-1 PHPoC file system

1.2 Path of Device Files

All device files of PHPoC are located in mmap (memory map) directory in root folder. To access to a specific device, you should use a path like the example below.

```
/mmap/DEVICE NAME
```

☞ ***ALL files that user upload to PHPoC are located in the root directory. Users can only access to the root and /mmap directory in this file system. In addition, users are not allowed to make or remove directories***

1.3 Types of Devices

PHPoC provides such types of devices below.

Hardware: Digital I/O port, UART(Serial) Port, NET(Ethernet or wireless LAN), Software: TCP, UDP, ST(Timer)

Types may different according to products and version of firmware.

☞ ***Refer to chapter 10 for more detailed device information about devices depending on the types of products.***

1.4 Steps of Using Devices

General steps of using devices are as follows:

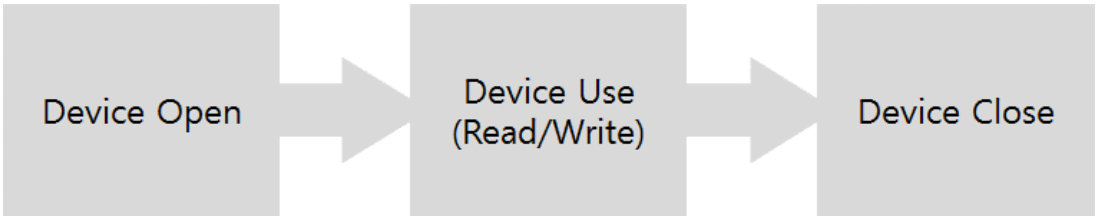


Figure 1-2 steps of using devices

1.4.1 Opening Device

pid_open function is for opening devices. This function returns pid (Peripheral ID), which is an integer value, and this value is used for accessing to devices as a unique number.

1.4.2 Using Device

After opening successfully, device, which returned pid indicates, is ready to use.

1.4.3 Closing Device

When device is not used anymore, close it by using pid_close function.

☞ ***pid_close function may not be required based on the types of devices.***

2 Digital I/O

2.1 Overview

Digital I/O can be used to monitor digital inputs or control digital outputs. This device is also used when connecting LED indicators which shows system status or setting types of serial communication.

- Digital I/O Structure

Every digital I/O port can have two different states which are High (or 1) and Low (or 0). Therefore, each port is matched to a binary digit as you can see below.

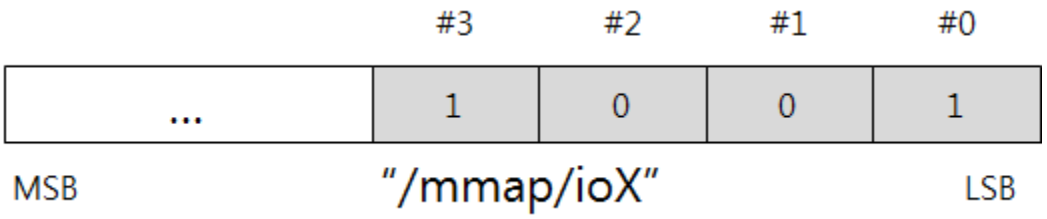


Figure 2-1 example of mapping digital I/O

2.2 Steps of Using Digital I/O

General steps of using digital I/O ports are as follows:

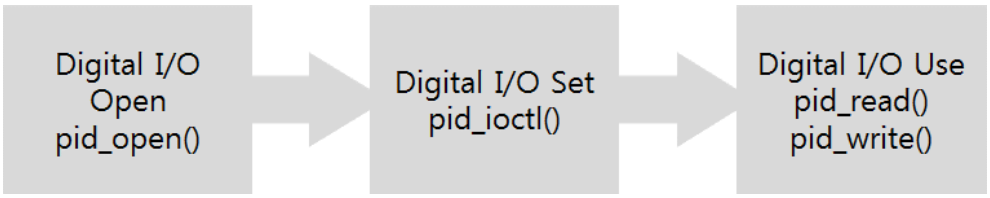


Figure 2-2 steps of using digital I/O

2.3 Opening Digital I/O

To open digital I/O, pid_open function is required.

```
$pid = pid_open("/mmap/io3");          // opening a 3rd digital I/O
```

☞ ***Including LED, refer to chapter 10 for detailed digital I/O information depending on the types of products.***

2.4 Setting Digital I/O

Before using digital I/O, setting each port is required. To set them, set command of pid_ioctl function is used.

```
pid_ioctl($pid, "set N1[-N2] TYPE");
```

N1 and N2 specifies a range of multiple ports. Users can omit the N2 in the case of setting a single port. Types defines use of the specified ports. Note that <space> should be between those two items.

2.4.1 Available Digital I/O Types

TYPE	Description
in	Input
out	Output
led_sys	System Status LED
led_net0_act / led_net1_act	Activation of NET(net0 - wired, net1 - wireless) link LED: - successfully established network link - LOW - at the moment sending data to network - HIGH - at the moment receiving data from network - HIGH
led_net0_link / led_net1_link	Network Link LED: connected to network - LOW
led_net0_rx / led_net1_rx	Network Receive LED: at the moment receiving data - LOW
led_net0_tx / led_net1_tx	Network Send LED: at the moment sending data - LOW

Table 2-1 available digital I/O types

☞ ***With products that equip LED, each port of digital I/O can be matched to physical LED. The LED is turned ON when the port state is LOW and turned OFF when it is HIGH. If you need more detailed information, please refer to circuit diagram of your product.***

- example of setting digital I/O

```
$pid = pid_open("/mmap/io3");          // opening digital I/O 3
pid_ioctl($pid, "set 0 out");          // setting port 0 to output
pid_ioctl($pid, "set 1-2 in");         // setting port 1 ~ 2 to input
pid_ioctl($pid, "set 3 led_net0_link");  // setting port 3 to network link LED
pid_ioctl($pid, "set 12 led_net0_rx");// setting port 12 to network receive LED
pid_ioctl($pid, "set 13 led_net0_tx");  // setting port 13 to network send LED
```

2.5 Using Digital I/O

2.5.1 Reading states of Digital I/O

To read status of digital I/O, pid_read function is required.

```
pid_read($pid, VALUE);
```

VALUE is an integer variable where you put read value in.

- example of reading states of digital I/O

The example below shows to print status of port 0 to 3 after setting them to input and getting the status.

```
$value = 0;
$pid = pid_open("/mmap/io3");          // opening digital I/O 3
pid_ioctl($pid, "set 0-3 in");         // setting port 0 ~ 3 to input
sleep(1);
pid_read($pid, $value);                // reading status
$value &= 0x0f;
printf("0x%02x\r\n", $value);          // result: e.g. 0x0f
```

2.5.2 Writing Values to Digital I/O

To write values, pid_write function is required.

```
pid_write($pid, VALUE);
```

VALUE is an integer variable for writing to digital I/O.

- example

  Following example prints states of digital I/O ports after setting 0 ~ 3 pins of digital I/O port 3 to output port and writing a given value.

```
$value = 0;
$pid = pid_open("/mmap/io3");          // opening digital I/O 3
pid_ioctl($pid, "set 0-3 out");        // setting port 0 ~ 3 to output
sleep(1);
pid_write($pid, 0x05);                 // writing 0x05
pid_read($pid, $value, 1);             // reading status of digital I/O
$value &= 0x0f;
printf("0x%02x\r\n", $value);          // result: 0x05
```

☞ ***In the case of reading or writing a specific port, proper bit operation is required because writing of digital I/O is implemented in WORD unit.***

2.5.3 Controlling Relay

Some external products have relay ports which connected to digital output.

- example

The example below simultaneously turns all relay output ports on and off every second after turning the relay port 0 to 3 on in order.

```
$pid = pid_open("/mmap/io4");          // opening digital I/O 4 (relay port)
pid_ioctl($pid, "set 7 out");          // Setting port 7(Relay OE) to output
pid_ioctl($pid, "set 8-11 out");       // Setting port 8 ~ 11 to output
                                       // (relay port 0 ~ 3)
pid_write($pid, 0x0100);               // turn on relay port 0
sleep(1);
pid_write($pid, 0x0200);               // turning relay port 1 on
sleep(1);
pid_write($pid, 0x0400);               // turning relay port 2 on
sleep(1);
pid_write($pid, 0x0800);               // turning relay port 3 on
sleep(1);
pid_write($pid, 0x0f00);               // turning all relay port on
sleep(1);
pid_write($pid, 0x0000);               // turning all relay port off
```

☞ ***This example is not available on products which do not have a relay port.***

3 UART

UART (Universal Asynchronous Receiver and Transmitter) is the most widely used serial communication in the world.

3.1 Steps of Using UART

General steps of using UART ports are as follows:

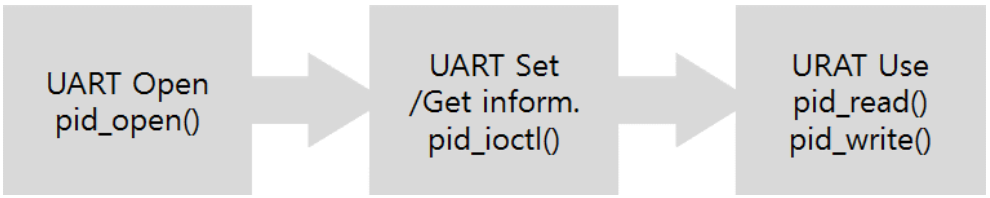


Figure 3-1 steps of using UART

3.2 Opening UART

To open UART, pid_open function is required.

```
$pid = pid_open("/mmap/uart0");                    // opening UART 0
```

☞ ***Refer to chapter 10.2.1 for detailed UART information depending on the types of products.***

3.3 Setting UART Port

Before using UART, setting parameters is required. To set them, set command of pid_ioctl function is used.

```
pid_ioctl($pid, "set ITEM VALUE");
```

ITEM means setting items and VALUE is possible value of the item.

3.3.1 Available UART items

ITEM	VALUE	Description
baud	e.g. 9600	baud rate[bps], 2400 ~ 230400
parity	0	no parity
	1	EVEN parity
	2	ODD parity
	3	MARK parity (always 1)
	4	SPACE parity (always 0)
data	8	8 data bit
	7	7 data bit(it can be only used with parity bit)
stop	1	1 stop bit
	2	2 stop bit
flowctrl	0	no flow control
	1	RTS/CTS hardware flow control
	2	Xon/Xoff software flow control
	3	TxDE flow control for RS485

Table 3-1 available UART items

- example of setting UART

```
$pid = pid_open("/mmap/uart0");      // opening UART 0
pid_ioctl($pid, "set baud 9600");    // baud rate > 9600 bps
pid_ioctl($pid, "set parity 0");     // no parity
pid_ioctl($pid, "set data 8");       // data bit length > 8
pid_ioctl($pid, "set stop 1");       // stop bit length > 1
pid_ioctl($pid, "set flowctrl 0");   // no flow control
```

3.3.2 Setting UART Communication Type

UART can be configured to RS422 or RS485 as well as RS232 depends on the types of products. For that, TxDE and setting related digital I/O are required.

This example shows how to set serial communication types of UART 0.

```
$pid = pid_open("/mmap/uart0");       // opening UART 0
pid_ioctl($pid, "set baud 9600");     // baud rate > 9600 bps
pid_ioctl($pid, "set parity 0");      // parity > none
pid_ioctl($pid, "set data 8");        // data bit > 8
pid_ioctl($pid, "set stop 1");        // stop bit > 1
pid_ioctl($pid, "set flowctrl 3");    // flow control > TxDE
$pid_mode = pid_open("/mmap/io4"); // opening Digital I/O 4 for UART mode
pid_ioctl($pid_mode, "set 0-3 out");  // setting port 0~3 to output
pid_write($pid_mode, 0x05);           // RS232
//pid_write($pid_mode, 0x02);         // (remove comment sign for RS422)
//pid_write($pid_mode, 0x0c);         // (remove comment sign for RS485)
pid_close($pid_mode);
```

☞ ***These examples assumed that pin 0~3 of digital I/O number 4 are for configuration of UART types. Please check device information because the digital I/O assignment may be different according to types of products.***

☞ ***Refer to chapter 10 for detailed digital I/O information depending on the types of products.***

3.4 Getting Status of UART Port

To get various states of UART, get command of pid_ioctl function is required.

```
$return = pid_ioctl($pid, "get ITEM");
```

3.4.1 Available UART States

ITEM	Description	Return Value	Return Type
baud	baud rate[bps]	e.g. 9600	Integer
parity	parity	0 / 1 / 2 / 3 / 4	Integer
data	data bit[bit]	8 / 7	Integer
stop	stop bit[bit]	1 / 2	Integer
flowctrl	flowctrl	0 / 1 / 2 / 3	Integer
txbuf	size of send buffer[Byte]	e.g. 1024	Integer
txfree	remaining send buffer size[Byte]	e.g. 1024	Integer
rxbuf	size of receive buffer[Byte]	e.g. 1024	Integer
rxlen	received data size[Byte]	e.g. 10	Integer
idle	receive idle time[ms]	e.g. 500	Integer

Table 3-2 available UART states

- example of getting UART states

Checking current information of UART port is as follows:

```
$pid = pid_open("/mmap/uart0");                    // opening UART 0
$baud = pid_ioctl($pid, "get baud");               // getting baud rate
$parity = pid_ioctl($pid, "get parity");           // getting parity
$data = pid_ioctl($pid, "get data");               // getting data bit
$stop = pid_ioctl($pid, "get stop");               // getting stop bit
$flowctrl = pid_ioctl($pid, "get flowctrl");       // getting flow control
echo "baud = $baud\r\n";                           // result e.g.: baud = 9600
echo "parity = $parity\r\n";                       // result e.g.: parity = 0
echo "data = $data\r\n";                           // result e.g.: data = 8
echo "stop = $stop\r\n";                           // result e.g.: stop = 1
echo "flowctrl = $flowctrl\r\n";                   // result e.g.: flowctrl = 0
```

3.4.2 Remaining Data Size in Send Buffer

Remaining data size in send buffer can be calculated as follows:

```
remaining data size in send buffer = size of buffer - remaining size of buffer
```

- example

This example shows how to check remaining data size in send buffer.

```
$data = "0123456789";
$pid = pid_open("/mmap/uart0");       // opening UART 0
pid_ioctl($pid, "set baud 9600");     // baud rate > 9600 bps
pid_ioctl($pid, "set parity 0");      // parity > none
pid_ioctl($pid, "set data 8");        // data bit > 8
pid_ioctl($pid, "set stop 1");        // stop bit > 1
pid_ioctl($pid, "set flowctrl 0");    // flow control > none
pid_write($pid, $data);               // writing data to UART port
$txbuf = pid_ioctl($pid, "get txbuf");   // getting send buffer size
$txfree = pid_ioctl($pid, "get txfree"); // getting remaining size of send buffer
$txlen = $txbuf - $txfree;        // getting remaining data size in send buffer
echo "tx len = $txlen\r\n";       // printing remaining data size in send buffer
```

3.4.3 Received Data Size

The following shows how to get received data size of UART port.

```
$rxlen = pid_ioctl($pid, "get rxlen[ $string]");
```

- Getting received data size with a string

When a string is specified after rxlen item, pid_ioctl function returns 0 until the string comes into UART. If the string comes, the function returns whole data size including the string.

3.4.4 Remaining Size of Receive Buffer

Remaining size of receive buffer can be calculated as follows:

```
remaining size of receive buffer = size of buffer - received data size
```

- example

This example shows how to get remaining size of receive buffer.

```
$rdata = "";
$pid = pid_open("/mmap/uart0");       // opening UART 0
pid_ioctl($pid, "set baud 9600");     // baud rate > 9600 bps
pid_ioctl($pid, "set parity 0");      // parity > none
pid_ioctl($pid, "set data 8");        // data bit > 8
pid_ioctl($pid, "set stop 1");        // stop bit > 1
pid_ioctl($pid, "set flowctrl 0");    // flow control > none
$rxbuf = pid_ioctl($pid, "get rxbuf");  // getting receive buffer size
$rxlen = pid_ioctl($pid, "get rxlen");  // getting received data size
$rxfree = $rxbuf - $rxlen;       // getting remaining size of receive buffer
echo "rxfree = $rxfree\r\n";     // printing remaining size of receive buffer
```

3.4.5 Receive Idle Time

Receive idle time means amount of time passed since the last data come to UART. The idle time can be confirmed as follows:

```
$rxidle = pid_ioctl($pid, "get idle");
```

The unit of returned value is millisecond.

3.5 Using UART

3.5.1 Reading Data

Received data from UART is stored in receive buffer. pid_read function is required to read the data.

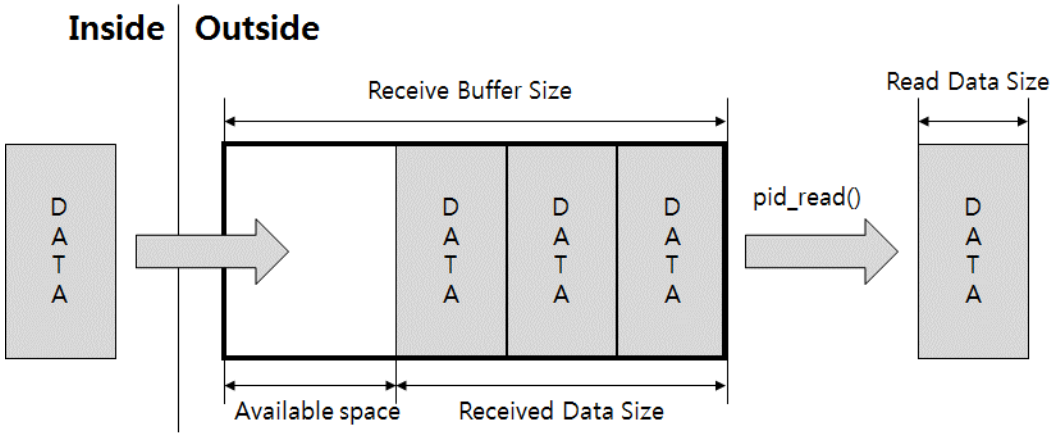


Figure 3-2 reading data from UART

The following shows how to use the pid_read function.

```
pid_read($pid, $var[, $len]);
```

Argument $var is a variable for saving the read data and $len is size of read data.

- Example

This example checks and prints received data to UART every second.

```
$rdata = "";
$pid = pid_open("/mmap/uart0");                 // opening UART 0
pid_ioctl($pid, "set baud 9600");               // baud rate > 9600bps
pid_ioctl($pid, "set parity 0");                // parity > none
pid_ioctl($pid, "set data 8");                  // data bit > 8
pid_ioctl($pid, "set stop 1");                  // stop bit > 1
$rxbuf = pid_ioctl($pid, "get rxbuf");          // getting size of receive buffer
while(1)
{
        $rxlen = pid_ioctl($pid, "get rxlen");  // getting size of received data
        $rx_free = $rxbuf - $rxlen;             // getting remaining size
        echo "$rx_free / $rxbuf\r\n";           // printing remaining size
        $len = pid_read($pid, $rdata, $rxlen); // reading data
        echo "len = $len / ";                   // printing size of read data
        echo "rdata = $rdata\r\n";              // printing read data
        sleep(1);
}
```

3.5.2 Sending Data

Data, written by pid_write function, is stored in send buffer and transferred to outside via UART.

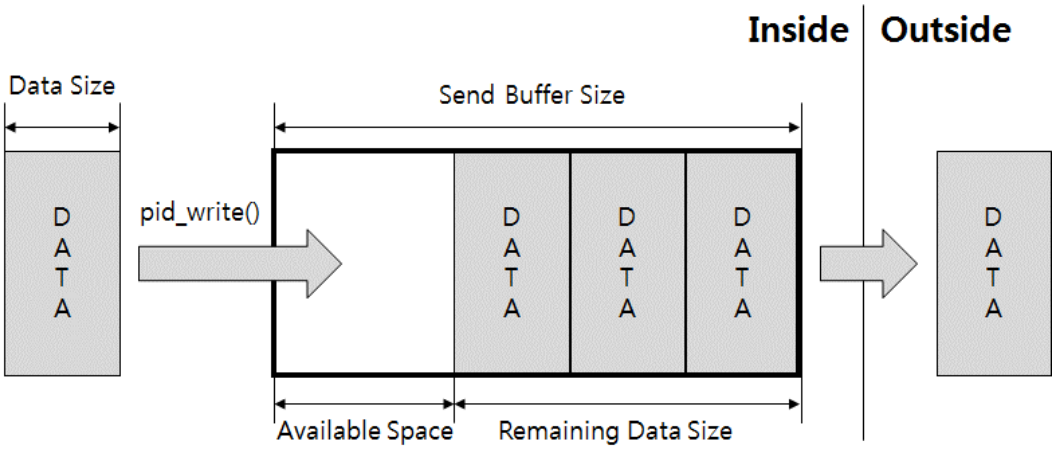


Figure 3-3 sending data to UART

The following shows how to use pid_write function.

```
pid_write($pid, $var[, $wlen]);
```

Argument $var is a variable for sending data and $wlen is a size of sending data.

- example

This example prints remaining size of send buffer and length of sent data every second.

```
$sdata = "0123456789";
$pid = pid_open("/mmap/uart0");                 // opening UART 0
pid_ioctl($pid, "set baud 9600");               // baud rate > 9600bps
$txbuf = pid_ioctl($pid, "get txbuf");          // getting size of send buffer
while(1)
{
        $txfree = pid_ioctl($pid, "get txfree"); // getting remaining size
        echo "txfree = $txfree\r\n";            // printing remaining size
        $len = pid_write($pid, $sdata, $txfree);// writing data
        echo "len = $len\r\n";                  // printing length of data sent
        sleep(1);
}
```

The third argument of pid_write function means the length of writing data. The length of writing data should be less than remaining size of send buffer to avoid data loss. It is highly recommended to check remaining size of send buffer before sending data.

4 NET (Network)

NET means physical wired and wireless network interface.

4.1 Steps of Using NET

General steps of using NET are as follows:

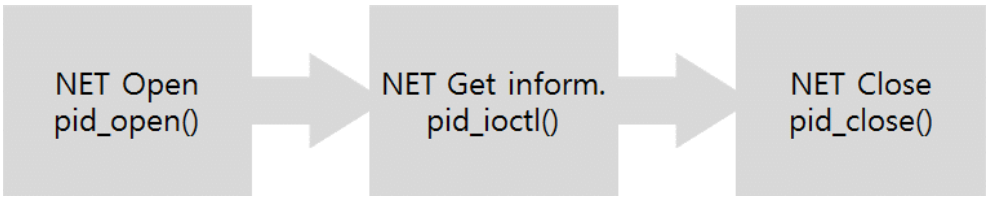


Figure 4-1 steps of using NET

4.2 Opening NET

To open NET, pid_open function is required.

```
$pid = pid_open("/mmap/net0");                    // opening NET 0
```

☞ ***Refer to chapter 10 for detailed NET information depending on the types of products.***

4.3 Getting Status of NET

To get a status of the NET port, get command of pid_ioctl function is required.

```
$return = pid_ioctl($pid, "get ITEM");
```

ITEM is a name of available states.

4.3.1 Available NET States

ITEM	Description	Return Value	Return Type
hwaddr	MAC Address	e.g. 00:30:f9:00:00:01	string
ipaddr	IP Address	e.g. 10.1.0.1	string
netmask	Subnet Mask	e.g. 255.0.0.0	string
gwaddr	Gateway Address	e.g. 10.1.0.254	string
nsaddr	Name Server Address	e.g. 10.1.0.254	string
mode	10M Ethernet	10BASET	string
	100M Ethernet	100BASET	string
	WLAN Unavailable	""(an Empty String)	string
	WLAN Infrastructure	INFRA	string
	WLAN Ad-hoc	IBSS	string
	WLAN Soft AP	AP	string
speed	Ethernet Speed[Mbps]	0 / 10 / 100	integer
	WLAN Speed[100Kbps]	0 / 10 / 20 / 55 / 110 / 60 / 90 / 120 / 180 / 240 / 360 / 480 / 540	integer

Table 4-1 available NET states

- example of getting NET states

This example checks and prints various states of NET.

```
$pid = pid_open("/mmap/net0");                  // opening NET 0
echo pid_ioctl($pid, "get hwaddr"), "\r\n";     // getting MAC address
echo pid_ioctl($pid, "get ipaddr"), "\r\n";     // getting IP address
echo pid_ioctl($pid, "get netmask"), "\r\n";    // getting subnet mask
echo pid_ioctl($pid, "get gwaddr"), "\r\n";     // getting gateway address
echo pid_ioctl($pid, "get nsaddr"), "\r\n";     // getting name server address
echo pid_ioctl($pid, "get mode"), "\r\n";       // getting Ethernet mode
echo pid_ioctl($pid, "get speed"), "\r\n";      // getting link speed
pid_close($pid);                                // closing NET 0
```

5 TCP

TCP is one of the most fundamental protocol along with UDP, in charge of transmission on TCP/IP, and it is widely used in present Internet. This protocol contains connection phase and provides data integrity.

5.1 Steps of Using TCP

General steps of using TCP are as follows:

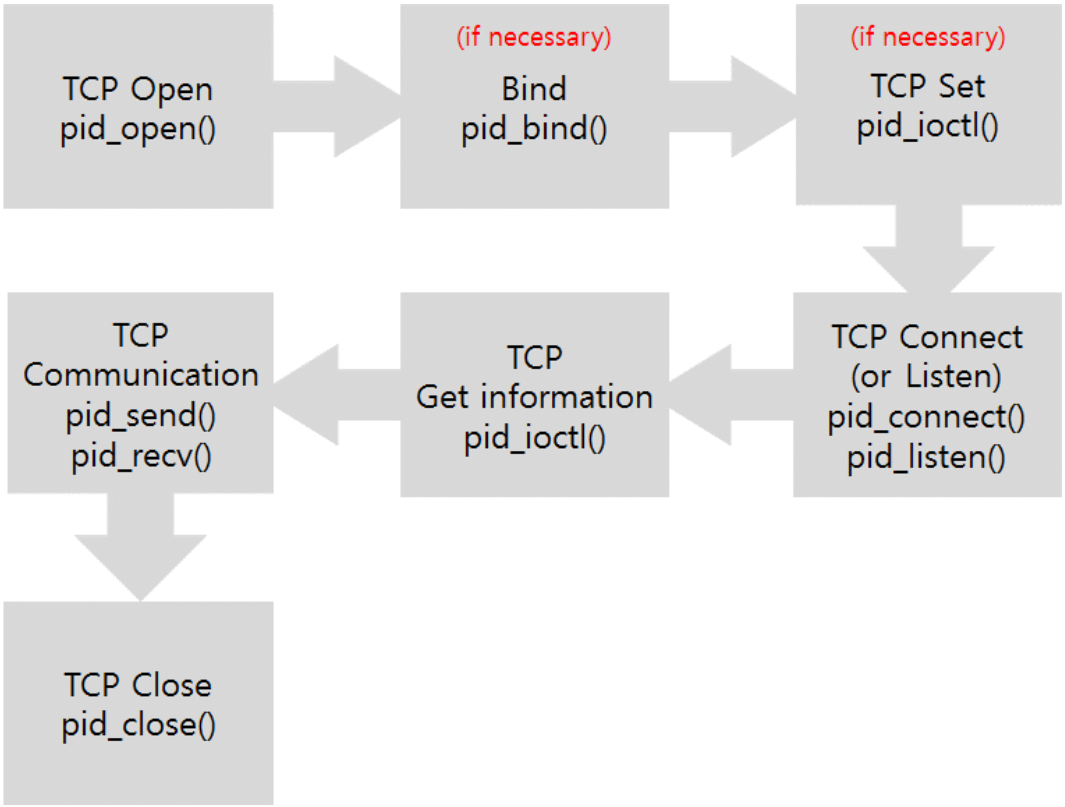


Figure 5-1 steps of using TCP

☞ ***In the case of using device to perform as TCP server, binding phase cannot be omitted.***

5.2 Opening TCP

To open TCP session, pid_open function is required.

```
$pid = pid_open("/mmap/tcp0");                    // opening TCP 0
```

☞ ***Refer to chapter 10 for detailed TCP information depending on the types of products.***

5.3 Setting TCP

Some parameters may be needed to set before using TCP. On SSL communication, especially, SSL setting is required before connection by set command of pid_ioctl function.

```
pid_ioctl($pid, "set ITEM VALUE");
```

ITEM means setting items and VALUE is possible value of the item.

5.3.1 Available TCP Items

ITEM	VALUE	Description
nodelay	0	Enable Nagle algorithm
	1	Disable Nagle algorithm
api	ssl	Use SSL
ssl method	ssl3_client	SSL client (SSL3.0)
	tls1_client	SSL client (TLS 1.0)
	ssl3_server	SSL server (SSL 3.0)
	tls1_server	SSL server (TLS 1.0)

Table 5-1 available TCP items

TCP Nagle Algorithm is to improve effective data transmission by reducing the number of segments. Thus, it may accompany a little delay.

- example of setting TCP items

```
$sdata = "01234567";
$pid = pid_open("/mmap/tcp0");                  // opening TCP
pid_ioctl($pid, "set nodelay 0");               // enabling Nagle algorithm
pid_ioctl($pid, "set api ssl");                 // using SSL
pid_ioctl($pid, "set ssl method tls1_client "); // setting SSL Client (TLS 1.0)
```

5.3.2 Notifications about SSL

- It is required to set API to SSL by "set api ssl" command before using SSL
- Certification must be saved in PHPoC when the device performs as SSL server.
- SSL communication could not be performed in case of lack of memory that caused by increased memory usage of PHPoC.

5.4 TCP Connection

5.4.1 TCP Client (Active Connection)

Active connection means sending TCP connection request packet to TCP server and this host is called TCP client. To perform TCP client, pid_connect function is used.

```
pid_connect($pid, $addr, $port);
```

Argument $addr is IP address of TCP server and $port is port number.

- example of TCP client

```
$pid = pid_open("/mmap/tcp0");      // opening TCP
$addr = "10.1.0.2";                 // IP address of TCP server
$port = 1470;                       // TCP port
pid_connect($pid, $addr, $port);    // active TCP connection
sleep(25);
```

5.4.2 TCP Server (Passive Connection)

Passive connection means listening TCP connection request packet from TCP client and this host is called TCP server. To perform TCP server, pid_bind and pid_listen function are required.

```
pid_bind($pid, "", $port);
pid_listen($pid[, $backlog]);
```

Argument $port is TCP port number.

- example of TCP Server

```
$pid = pid_open("/mmap/tcp0");      // opening TCP
$port = 1470;                       // TCP port number
pid_bind($pid, "", $port);          // binding with the port number
pid_listen($pid);                   // passive TCP connection
sleep(25);
```

5.5 Getting Status of TCP

To get states of TCP, get command of pid_ioctl function is required.

```
$return = pid_ioctl($pid, "get ITEM");
```

5.5.1 Available TCP States

ITEM	Description	Return Value	Return Type
state	TCP session is closed	TCP_CLOSED	integer
	TCP session is connected	TCP_CONNECTED	integer
	TCP session waits for connection	TCP_LISTEN	integer
	SSL session is closed	SSL_CLOSED	integer
	SSL session is connected	SSL_CONNECTED	integer
	SSL session waits for connection	SSL_LISTEN	integer
srcaddr	local IP address	e.g. 192.168.0.1	string
srcport	local port number	e.g. 1470	integer
dstaddr	peer IP address	e.g. 192.168.0.2	string
dstport	peer TCP number	e.g. 1470	integer
txbuf	size of send buffer[Byte]	e.g. 1152	integer
txfree	remaining send buffer size[Byte]	e.g. 1152	integer
rxbuf	size of receive buffer[Byte]	e.g. 1068	integer
rxlen	received data size[Byte]	e.g. 200	integer

Table 5-2 available TCP states

5.5.2 TCP Session Status

Checking status of connection on TCP is very important, because TCP data communication is made after the connection phase. There are three session states: TCP_CLOSED when session has not been connected, TCP_CONNECTED when session has been connected and TCP_LISETN when TCP server has been listening connection. SSL communication has also these three states: SSL_CLOSED, SSL_CONNECTED and SSL_LISTEN. The following shows how to get states of session.

```
$state = pid_ioctl($pid, "get state");
```

5.5.3 Remaining Data Size in Send Buffer

Remaining data size in send buffer can be calculated as follows:

```
remaining data size in send buffer = size of buffer - remaining size of buffer
```

- example

This example shows how to get remaining data size in send buffer.

```
$pid = pid_open("/mmap/tcp0");          // opening TCP
pid_connect($pid, "10.1.0.2", 1470);    // TCP active connection
$txbuf = pid_ioctl($pid, "get txbuf");  // getting send buffer size
$txfree = pid_ioctl($pid, "get txfree"); // getting remaining size of send buffer
$tx_len = $txbuf - $txfree;             // getting remaining data size
echo "tx len = $tx_len\r\n";            // printing remaining data size
sleep(5);
pid_close($pid);                        // closing TCP
```

5.5.4 Received Data Size

The following shows how to get received data size from TCP.

```
$rxlen = pid_ioctl($pid, "get rxlen[ $string]");
```

- Getting received data size with string

When a string is specified after rxlen item, pid_ioctl function returns 0 until the string comes to serial port. If the string comes, the function returns whole data size including the string.

5.5.5 Remaining Size of Receive Buffer

Remaining size of receive buffer can be calculated as follows:

```
remaining size of receive buffer = size of buffer - size of received data
```

- example

This example shows how to get remaining size of receive buffer.

```
$pid = pid_open("/mmap/tcp0");          // opening TCP
pid_connect($pid, "10.1.0.2", 1470);    // TCP active connection
$rxbuf = pid_ioctl($pid, "get rxbuf");  // getting receive buffer size
$rxlen = pid_ioctl($pid, "get rxlen");  // getting received data size
$rx_free = $rxbuf - $rxlen;             // getting remaining size
echo "rx free = $rx_free\r\n";          // printing remaining size
sleep(5);
pid_close($pid);                        // closing TCP
```

5.6 TCP Communication

5.6.1 Receiving TCP Data

Received data from network via TCP is stored in receive buffer. pid_recv function is required to read the data.

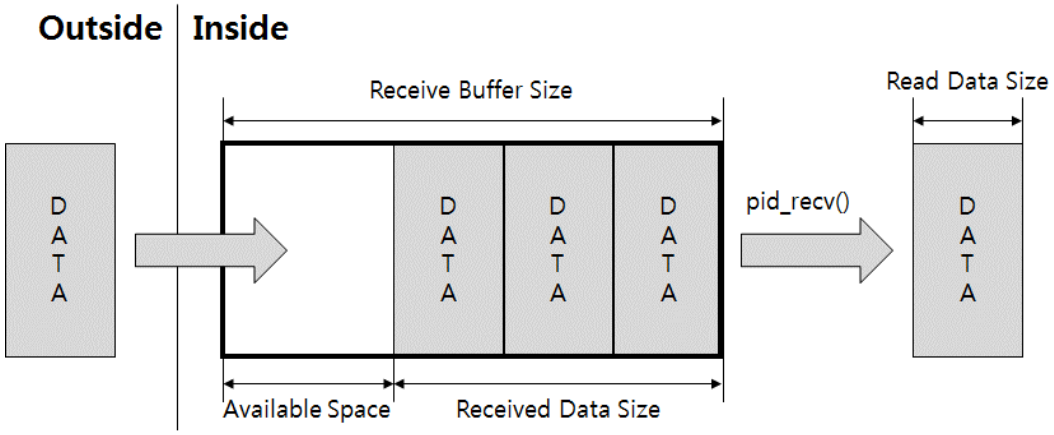


Figure 5-2 receiving TCP data

The following shows how to use pid_recv function.

```
pid_recv($pid, $value[, $len]);
```

- example

This example checks and prints received TCP data every second.

```
$rdata = "";
$pid = pid_open("/mmap/tcp0");                  // opening TCP
pid_connect($pid, "10.1.0.2", 1470);            // TCP active connection
do
{
        sleep(1);
        $state = pid_ioctl($pid, "get state");  // getting TCP session state
        $rxlen = pid_ioctl($pid, "get rxlen");  // getting received data size
        $rlen = pid_recv($pid, $rdata, $rxlen); // receiving data
        echo "rlen = $rlen / ";                 // printing received data size
        echo "rdata = $rdata\r\n";              // printing received data
}
while($state == TCP_CONNECTED);
```

5.6.2 Sending TCP Data

Sent data by pid_send function is stored in send buffer and transferred to network via TCP.

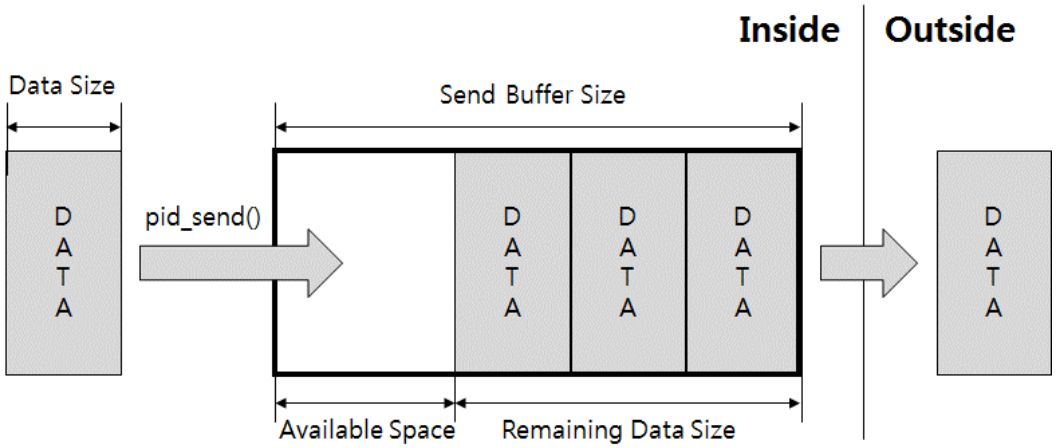


Figure 5-3 sending TCP data

The following shows how to use pid_send function.

```
pid_send($pid, $value[, $len]);
```

- example

This example sends data to network via TCP checking available space of send buffer every second.

```
$sdata = "0123456789";
$pid = pid_open("/mmap/tcp0");                  // opening TCP
pid_connect($pid, "10.1.0.2", 1470);            // TCP connection
do
{
        sleep(1);
        $state = pid_ioctl($pid, "get state");    // getting session state
        $txfree = pid_ioctl($pid, "get txfree"); // getting remaining size
        $tx_len = pid_send($pid, $sdata, $txfree);      // sending data
        echo "tx len = $tx_len\r\n";              // printing sent data size
}
while($state == TCP_CONNECTED);
```

The third argument of pid_write function means the length of sending data. The length of sending data should be less than remaining size of send buffer to avoid data loss. It is highly recommended to check remaining size of send buffer before sending data.

6 UDP

Although UDP does not offers data integrity and connection phase, it has good features such as simple header and prompt data transmission.

6.1 Steps of Using UDP

General steps of using UDP are as follows:

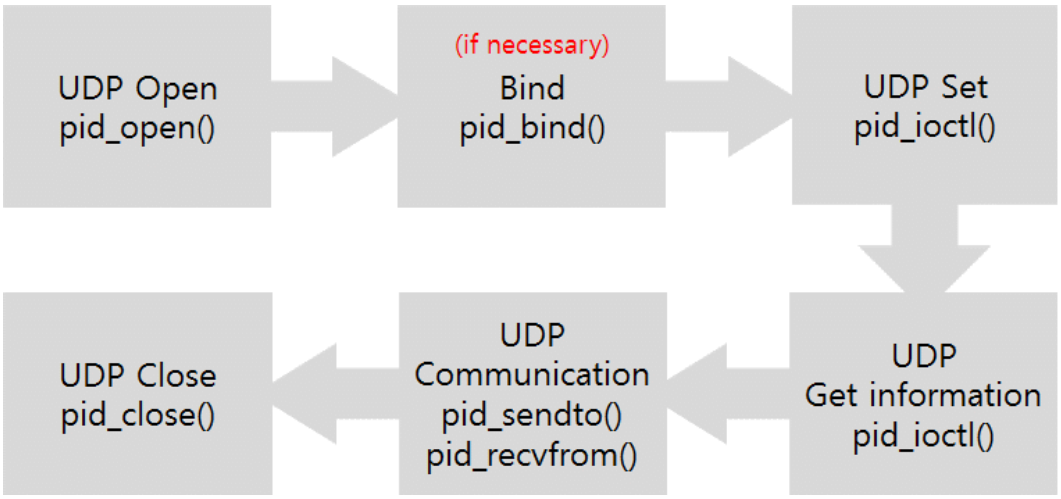


Figure 6-1 steps of using UDP

☞ ***Binding socket can be omitted if there is no requirement of prior setting or data transmission.***

6.2 Opening UDP

To open UDP, pid_open function is required.

```
$pid = pid_open("/mmap/udp0");        // opening UDP 0
```

☞ ***Refer to chapter 10 for detailed UDP information depending on the types of products.***

6.3 Binding

Binding which uses pid_bind function is required to specify destination IP address and to receive data from network via UDP.

```
$pid = pid_bind($pid, $addr, $port);
```

Argument $addr is IP address and $port is port number to bind. When the IP address specified empty string (""), PHPoC assume the value is the current local IP address.

☞ ***Empty string ("") value is the only option for $addr argument of function bind.***

- example of binding

```
$pid = pid_open("/mmap/udp0");      // opening UDP
$port = 1470;                       // UDP port number
pid_bind($pid, "", $port);          // binding
```

6.4 Setting UDP

Destination IP address and port number can be specified before sending UDP data. This feature allows to omit fourth and fifth argument of pid_send function.

Set command of pid_ioctl function is required to set UDP.

```
pid_ioctl($pid, "set ITEM VALUE");
```

ITEM means setting items and VALUE is possible value of the item.

6.4.1 Available UDP Items

ITEM	VALUE	Description
dstaddr	e.g. 10.1.0.2	destination IP address
dstport	e.g. 1470	destination port number

Table 6-1 available UDP items

- example of setting UDP

```
$pid = pid_open("/mmap/udp0");                  // opening UDP
pid_bind($pid, "", 1470);                       // binding
pid_ioctl($pid, "set dstaddr 10.1.0.2");        // destination IP address
pid_ioctl($pid, "set dstport 1470");            // destination port number
```

6.5 Getting UDP status

To get status of UDP, get command of pid_ioctl function is required.

```
$return = pid_ioctl($pid, "get ITEM");
```

6.5.1 Available UDP states

ITEM	Description	Return Value	Return Type
srcaddr	source IP address	e.g. 192.168.0.1	string
srcport	source port number	e.g. 1470	integer
dstaddr	destination IP address	e.g. 192.168.0.2	string
dstport	destination port number	e.g. 1470	integer
rxlen	received data size[Byte]	e.g. 200	integer

Table 6-2 available UDP states

6.5.2 Received Data Size

To get received data size, "get rxlen" command of pid_ioctl function is required.

```
$rxlen = pid_ioctl($pid, "get rxlen");
```

- example

This example quits after printing received data size if data comes from network while checking periodically whether there is data or not.

```
$rbuf = "";
$pid = pid_open("/mmap/udp0");                  // opening UDP
pid_bind($pid, "", 1470);                       // binding
do
{
        $rxlen = pid_ioctl($pid, "get rxlen");  // getting received data size
        if($rxlen)
        {
                pid_recvfrom($pid, $rbuf, $rxlen);      // receiving data
                echo "$rxlen bytes\r\n";         // printing received data size
        }
        usleep(100000);
}while($rxlen == 0)
```

6.6 UDP Communication

6.6.1 Receiving UDP Data

To receive data from network via UDP, pid_recvfrom function is required. There are two receive buffer of UDP and the following shows how they works.

☞ ***Refer to chapter 11 for information about UDP receive buffer size depending on the types of products.***

- receiving UDP data from network

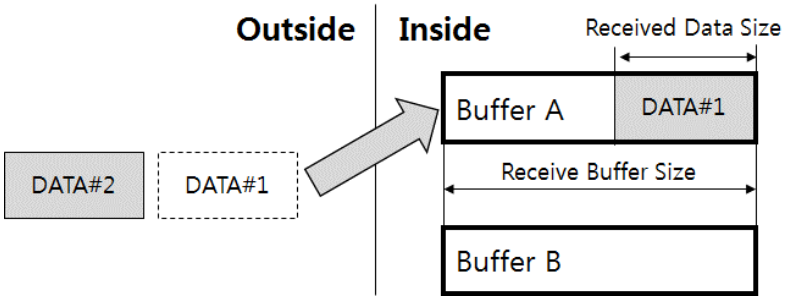


Figure 6-2 receiving UDP data from network

- reading UDP data from receive buffer

  After reading data from receive buffer by calling pid_recvfrom function, PHPoC flush the buffer.

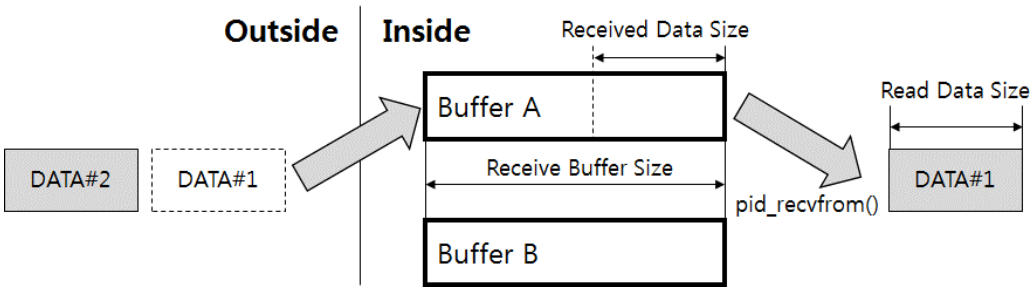


Figure 6-3 reading UDP data from receive buffer

- reading data size less than received data size

  Remaining data after reading will be lost by flushing receive buffer.

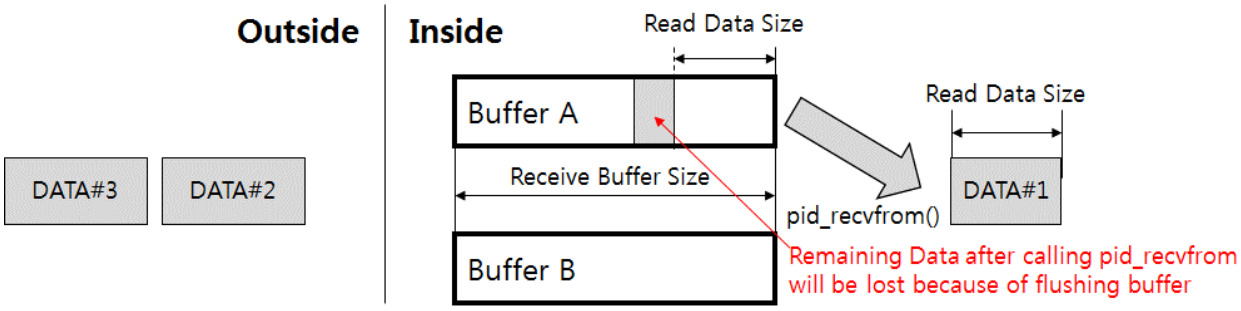


Figure 6-4 reading less size of data than received data size

- losing data by no available receive buffer

  While each of two receive buffers has data which is not read, subsequent data from network cannot be received. Therefore, it is recommended that immediately read data in received buffer right after checking received data size.

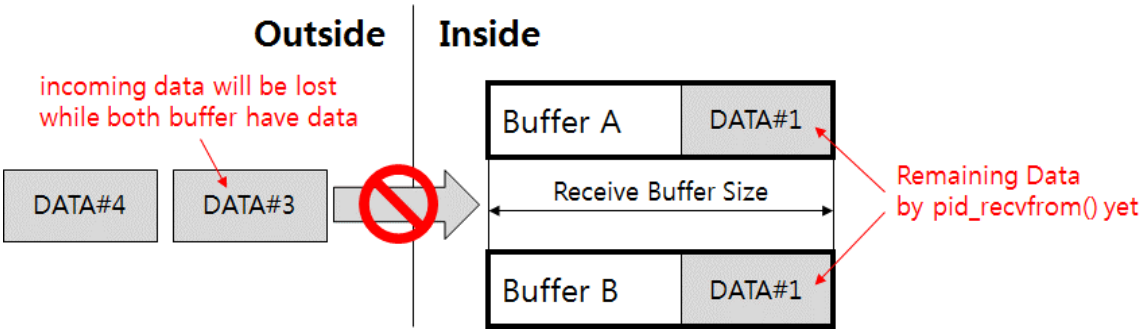


Figure 6-5 losing data by no available receive buffer

- example

This example prints received UDP data checking if there is data comes from network every second.

```
$rbuf = "";
$pid = pid_open("/mmap/udp0");                  // opening UDP
pid_bind($pid, "", 1470);                       // binding
do
{
        $rxlen = pid_ioctl($pid, "get rxlen");  // getting received data size
        if($rxlen)
        {
                pid_recvfrom($pid, $rbuf, $rxlen);      // receiving data
                echo "$rbuf\r\n";               // printing received data
        }
        usleep(100000);
}while(1)                                       // infinite loop
```

6.6.2 Sending UDP Data

To send UDP data, pid_sendto function is required.

- example of sending UDP data

```
$sdata = "01234567";
$pid = pid_open("/mmap/udp0");                                  // opening UDP
$slen = pid_sendto($pid, $sdata, 8, 0, "10.1.0.2", 1470);       // sending data
echo "slen = $slen\r\n";         // printing sent data size
```

7 ST (Timer)

ST (Timer) of PHP provides millisecond unit timer functions.

7.1 Steps of Using ST

General steps of using ST are as follows:

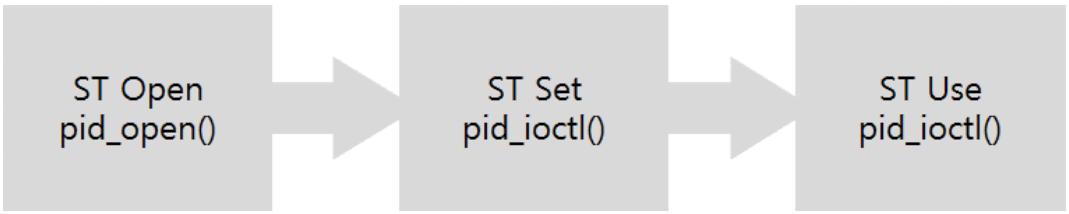


Figure 7-1 steps of using ST

7.2 Opening ST

To open ST, pid_open function is required.

```
$pid = pid_open("/mmap/st0");          // opening ST
```

☞ ***Refer to chapter 10 for detailed ST information depending on the types of products.***

7.3 Setting ST

ST allows to set direction (increment / decrement) and initial value by set command.

```
pid_ioctl($pid, "set ITEM VALUE");
```

ITEM means setting items and VALUE is possible value of the item.

7.3.1 Available ST Items

ITEM	VALUE	Description
count	e.g.) 5000	Initializing Timer Value(Unit: millisecond)
dir	up	Direction: increase(default value)
	down	Direction: decrease

Table 7-1 available ST items

7.4 Using ST

ST is started by "start" command and stopped by "stop" command of pid_ioctl function.

```
pid_ioctl($pid, "start");          // ST start
pid_ioctl($pid, "stop");           // ST stop
```

7.4.1 UP Count

ST value can be read by "get count" command of pid_ioctl function. The value means elapsed time from timer start. Unit of this value is millisecond.

```
$timer = pid_ioctl ($pid, "get count");
```

This example gets and prints ST value approximately every second after setting direction of ST to increment.

```
$pid = pid_open("/mmap/st0");                  // open ST
pid_ioctl($pid, "set dir up");                 // set ST direction to increase
pid_ioctl($pid, "start");                      // start ST
while(1)
{
        $value = pid_ioctl($pid, "get count"); // get ST value
        echo "$value\r\n";                     // print ST value
        sleep(1);
}
```

7.4.2 DOWN Count

This example gets and prints ST value approximately every second after setting direction of ST to decrement with 10 seconds initial value.

```
$pid = pid_open("/mmap/st0");                  // open ST
pid_ioctl($pid, "set dir down");               // set ST direction to decrease
pid_ioctl($pid, "set count 10000");            // set ST value to 10000
pid_ioctl($pid, "start");                      // start ST
for($i = 0; $i < 10; $i++)
{
        $value = pid_ioctl($pid, "get count"); // get ST value
        echo "$value\r\n";                     // print ST value
        sleep(1);
}
```

☞ ***In case of up direction, setting by "set count" will be ignored and the count value will be initialized to zero.***

8 Device Related Functions

PHPoC provides a bunch of internal functions for using devices as follows:

Function	Use Format
pid_bind	pid_bind(PID[, IP, PORT]);
pid_close	pid_close(FILE);
pid_connect	pid_connect(PID, IP, PORT);
pid_ioctl	pid_ioctl(PID, COMMAND);
pid_listen	pid_listen(PID, [BACKLOG]);
pid_open	pid_open(FILE);
pid_read	pid_read(PID, BUF[, LEN]);
pid_recv	pid_recv(PID, BUF[, LEN, FLAG]);
pid_recvfrom	pid_recvfrom(PID, BUF[, LEN, FLAG, IP, PORT]);
pid_send	pid_send(PID, BUF[, LEN, FLAG]);
pid_sendto	pid_sendto(PID, BUF[, LEN, FLAG, IP, PORT]);
pid_write	pid_write(PID, BUF[, LEN]);

Table 8-1 device related functions

☞ ***Refer to the Internal Functions Manual for detailed information of internal functions.***

9 Web Interface

9.1 Overview

PHPoC always provides web interface regardless of implementing script. Port number of web interface is TCP 80 and users can access the web page via web browser such as Internet Explorer, Chrome or etc.

9.2 Steps of Using Web Interface

9.2.1 Uploading Files for Web Page

Upload "index.php" file contains contents below to file system of PHPoC.

```
<html>
<body>
Hello PHPoC
</body>
</html>
```

9.2.2 Accessing to Web Page

Access to the web page entering IP address of PHPoC on the web browser.

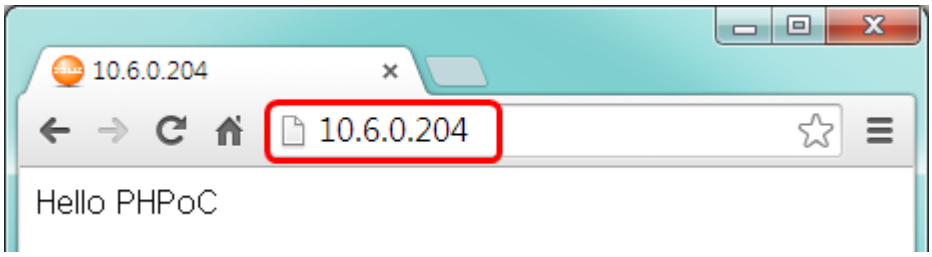


Figure 9-1 accessing to web page (1)

In the case that file name is not "index.php", path of the file including file name should be entered as you can see below.

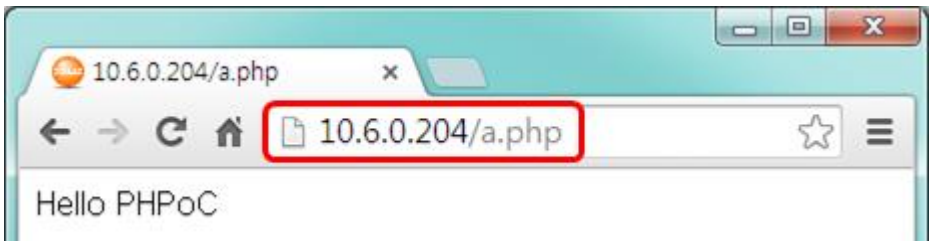


Figure 9-2 accessing to web page (2)

10 Device Information

10.1 The Number of Device Depending on Product Types

Device		PBH-101	PBH-104	PBH-204
UART		1	4	1
NET		2	2	2
TCP		5	5	5
UDP		5	5	5
I/O	Digital Input	0	0	4
	Digital Output	0	0	4
	Digital Output(LED)	8	8	8
ST		4	4	4

Table 10-1 the number of device depending on product types

10.2 Device File Path Depending on Product Types

10.2.1 UART

Product	Path of File
PBH-101, PBH-204	/mmap/uart0
PBH-104	/mmap/uart0
	/mmap/uart1
	/mmap/uart2
	/mmap/uart3

Table 10-2 UART

10.2.2 NET

Product	Path of File	Note
PBH-101, PBH-104, PBH-204	/mmap/net0	Wired LAN
	/mmap/net1	Wireless LAN

Table 10-3 NET

10.2.3 TCP

Product	Path of File
PBH-101, PBH-104, PBH-204	/mmap/tcp0
	/mmap/tcp1
	/mmap/tcp2
	/mmap/tcp3
	/mmap/tcp4

Table 10-4 TCP

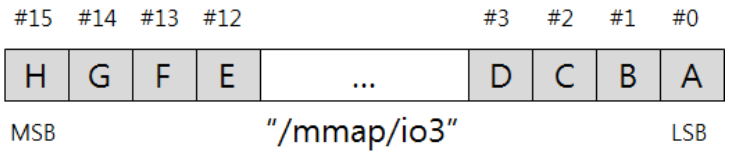
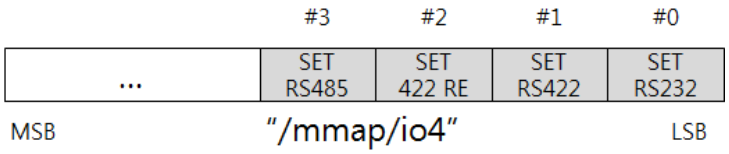
10.2.4 UDP

Product	Path of File
PBH-101, PBH-104, PBH-204	/mmap/udp0
	/mmap/udp1
	/mmap/udp2
	/mmap/udp3
	/mmap/udp4

Table 10-5 UDP

10.2.5 I/O

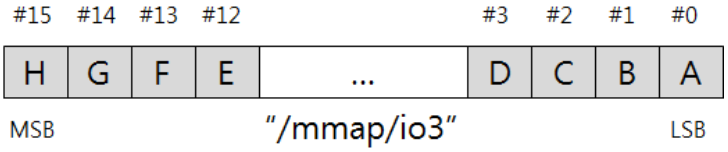
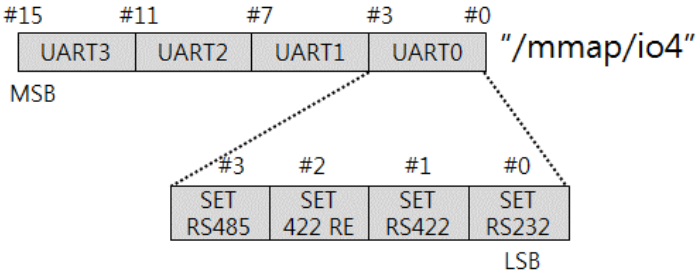
- PBH-101

<table>
<tr><th colspan="2">Division</th><th>Path of File and Mapping Information</th></tr>
<tr><td rowspan="2">PBH-101</td><td>LED</td><td>/mmap/io3
 
</td></tr>
<tr><td>UART Mode
(Serial Type)</td><td>/mmap/io4
 

● example of setting UART mode
<table>
<tr><th>Mode</th><th>Value</th><th>SET RS485</th><th>SET 422 RE</th><th>SET RS422</th><th>SET RS232</th></tr>
<tr><td>RS232</td><td>0x05</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>RS422</td><td>0x02</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>RS485</td><td>0x0c</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
</table></td></tr>
</table>

Table 10-6 digital I/O of PBH-101

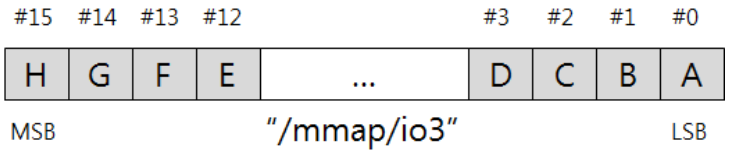
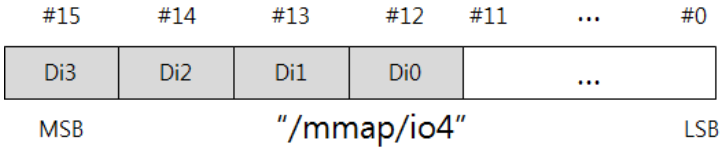
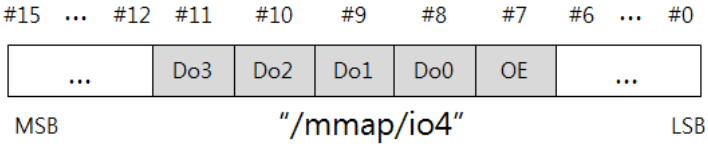
- PBH-104

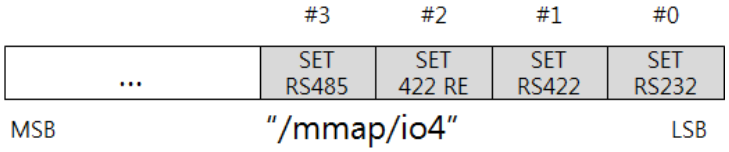
<table>
<tr><th colspan="2">Division</th><th>Path of File and Mapping Information</th></tr>
<tr><td rowspan="2">PBH-104</td><td>LED</td><td>/mmap/io3
 
</td></tr>
<tr><td>UART Mode
(Serial Type)</td><td>/mmap/io4
 

● example of setting UART mode
<table>
<tr><th>Mode</th><th>Value</th><th>SET RS485</th><th>SET 422 RE</th><th>SET RS422</th><th>SET RS232</th></tr>
<tr><td>RS232</td><td>0x05</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>RS422</td><td>0x02</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>RS485</td><td>0x0c</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
</table>
● example of setting multiple UART
<table>
<tr><th>Mode</th><th>Value</th><th>UART0</th><th>UART1</th><th>UART2</th><th>UART3</th></tr>
<tr><td>RS232</td><td>0x5555</td><td>0x0005</td><td>0x0050</td><td>0x0500</td><td>0x5000</td></tr>
<tr><td>RS422</td><td>0x2222</td><td>0x0002</td><td>0x0020</td><td>0x0200</td><td>0x2000</td></tr>
<tr><td>RS485</td><td>0xCCCC</td><td>0x000C</td><td>0x00C0</td><td>0x0C00</td><td>0xC000</td></tr>
</table></td></tr>
</table>

Table 10-7 digital I/O of PBH-104

- PBH-204

<table>
<tr><th colspan="2">Division</th><th>Path of File and Mapping Information</th></tr>
<tr><td rowspan="3">PBH-204</td><td>LED</td><td>/mmap/io3
 
</td></tr>
<tr><td>Digital Input (photo-coupler)</td><td>/mmap/io4
 
</td></tr>
<tr><td>Digital Output (relay)</td><td>/mmap/io4
 

※ OE: bit for enabling or disabling output relay
- Enable: LOW(0), Disable: HIGH(1)</td></tr>
<tr><td>UART Mode</td><td>/mmap/io4
 

● example of setting UART mode
<table>
<tr><th>Mode</th><th>Value</th><th>SET RS485</th><th>SET 422 RE</th><th>SET RS422</th><th>SET RS232</th></tr>
<tr><td>RS232</td><td>0x05</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>RS422</td><td>0x02</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td>RS485</td><td>0x0c</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
</table></td></tr>
</table>

Table 10-8 digital I/O of PBH-204

10.2.6 ST

Product	Path of File
PBH-101, PBH-104, PBH-204	/mmap/st0
	/mmap/st1
	/mmap/st2
	/mmap/st3

Table 10-9 ST

10.2.7 ENV and User Memory

Division		Path of File	Size (Byte)
PBH-101, PBH-104, PBH-204	System ENV	/mmap/envs	1520
	User ENV	/mmap/envu	1520
	User Memory	/mmap/um0	64
		/mmap/um1	64
		/mmap/um2	64
		/mmap/um3	64

Table 10-10 ENV and user memory

11 F/W Specification and Restriction

11.1 Firmware

Firmware	Product
P10	None
P20	PBH-101, PBH-104, PBH-204

Table 11-1 Firmware

11.2 Specification

Specification	Firmware			Description
	P10	P20	P30	
ENVS	1,024	1,536		Size of System ENV, byte
ENVU	1,024	1,536		Size of User ENV, byte
WLAN	X	O		Wireless LAN
EMAC	1			Ethernet
UART	1	4		The number of UART
FLOAT	X	O		Floating Point Numbers
SSL	X	O		SSL communication
PHP_MAX_NAME_SPACE	16			The number of Namespace
PHP_NAME_LEN	32			Size of User Identifier
PHP_MAX_USER_DEF_NAME	128	480		The number of User Identifier
PHP_LLSTR_BLK_SIZE	64			Size of String Block, byte
PHP_MAX_LLSTR_BLK	96	192		The number of String Blocks
string buffer size	6K	12K		Size of string buffer, byte
PHP_MAX_STRING_LEN	1K	1.5K		Size of string variable, byte
PHP_INT_MAX	about $9.2*10^{18}$			Max value of integer type
EZFS_MAX_NAME_LEN	64			Size of EZFS filename, byte
TASK	2			The number of Task
TCP	2	5		The number of TCP
UDP	2	5		The number of UDP
TCP_RXBUF_SIZE	1,024	1,068		TCP receive buffer size
TCP_TXBUF_SIZE	1,024	1,152		TCP send buffer size
PDB_TXBUF_SIZE	1,024	1,536		PHPoCD send buffer size
HTTP_TXBUF_SIZE	1,024	1,536		HTTP send buffer size
UART_RXBUF_SIZE	1,024			UART send/receive buffer size
UDP_RXBUF_SIZE	512			UDP receive buffer size
ADC	X	X	4	Analog to Digital Input
HT	X	X	4	Hardware Timer
SPI	X	X	1	SPI
I2C	X	X	1	I2C
RTC	X	X	O	RTC

Table 11-2 Firmware specification

11.3 Limitations

Item	limitation
Level of Namespace	PHP_MAX_NAME_SPACE - 1
Level of Function Call	PHP_MAX_NAME_SPACE - 2
Size of User Identifier	PHP_NAME_LEN - 1
Size of String Variable	PHP_MAX_STRING_LEN - 2
Size of Array Offset	string length - 2
Size of Filename	EZFS_MAX_NAME_LEN - 1
Size of arguments for system function	PHP_LLSTR_BLK_SIZE - 1
Size of arguments for pid_ioctl function	PHP_LLSTR_BLK_SIZE - 1
Size of system("pdkdf2") password & salt	PHP_LLSTR_BLK_SIZE - 1
Size of $address of function sendto	PHP_LLSTR_BLK_SIZE - 1
Size of $needle & $replace of function str_replace	PHP_LLSTR_BLK_SIZE - 1
Size of $address of function inet_pton	PHP_LLSTR_BLK_SIZE - 1
Size of $address of function inet_ntop	PHP_LLSTR_BLK_SIZE - 1
Size of $delimiter of function explode	PHP_LLSTR_BLK_SIZE - 1
Maximum size of UDP data for receiving	UDP receive buffer size - 2

Table 11-3 limitations

12 pid_ioctl Command Index

Command	Value	Device	Page
set	api	TCP	- 21 -
	baud	UART	- 11 -
	count	ST	- 32 -
	data	UART	- 11 -
	dir	ST	- 32 -
	dstaddr	UDP	- 28 -
	dstport	UDP	- 28 -
	flowctrl	UART	- 11 -
	N[-N]	I/O	- 7 -
	nodelay	TCP	- 21 -
	parity	UART	- 11 -
	ssl	TCP	- 21 -
	stop	UART	- 11 -
get	baud	UART	- 13 -
	count	ST	- 33 -
	data	UART	- 13 -
	dstaddr	TCP, UDP	- 23 -, - 29 -
	dstport	TCP, UDP	- 23 -
	flowctrl	UART	- 13 -
	gwaddr	NET	- 19 -
	hwaddr	NET	- 19 -
	ipaddr	NET	- 19 -
	mode	NET	- 19 -
	netmask	NET	- 19 -
	nsaddr	NET	- 19 -
	parity	UART	- 13 -
	rxbuf	UART, TCP	- 13 -, - 23 -
	rxlen	UART, TCP, UDP	- 13 -, - 23 -, - 29 -
	speed	NET	- 19 -
	srcaddr	TCP, UDP	- 23 -, - 29 -
	srcport	TCP, UDP	- 23 -, - 29 -
	state	TCP	- 23 -
	stop	UART	- 13 -
	txbuf	UART, TCP	- 13 -, - 23 -
	txfree	UART, TCP	- 13 -, - 23 -

Table 12-1 pid_ioctl Command Index

13 Revision History

Date	Version	Note	Author
2014.09.23	1.0	○ Create	Roy LEE