Ezi-IO® Plus-E

User Manual

Communication Function

(**Rev.01**)

FASTECH

Table of Contents

1 . Communication Protocol

1 - 1 . Communication Function

Ezi-IO Plus-E products can control up to 254 (1 ~ 254) modules through using Ethernet communication

1 - 1 - 1 . Communication Specifications

Item	Specification
Communication Speed	10/100base-T/TX
Communication Type(Protocol)	UDP (Port No. : **3001**)
Max Cabling Length	100m
Min Cable length between drive	More than 20cm
Number of Connected Axes	254axes (No. 01~FE)

1 - 1 - 2 . Ethernet IP address

1) Subnet Mask : 255.255.255.0
2) Gateway : 192.168.0.1
3) IP address : 192.168.0.xxx (xxx is set by an external switch)

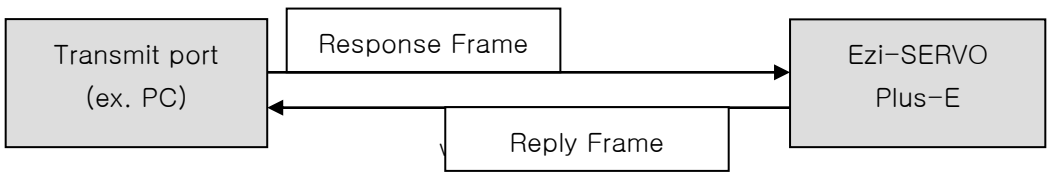
- When connecting to Ezi-SERVOⅡ Plus-E directly from a PC or Ethernet device, be sure to set the network setting according to the above IP address.
  If it is not set or is different, is can not be connected.
- If the switch set to 255(FF), IP address is automatically set.

Because it uses DHCP, IP address set automatically only when using router.

- When connecting directly from the controller(PC, PLC, etc.), be sure to set ther IP address with the switch
- Set the IP address automatically only when the default IP address is not used.
  If the IP set automatically, connect the user program (GUI), save the IP address, turn off the power, and set the last number of the IP with the switch.
- When the IP setting switch set to 0, the IP setting is reset to the above value.

1 - 1 - 3 . Ethernet Protocol

1) Overview of communication FRAME



2) Basic structure of FRAME

UDP Header	*Frame Data*
8bytes	5~257 bytes

The UDP Header contains the following information:

① Transmit port numberSending : 2bytes

② Receiving port number : 2bytes

③ Data length : 2bytes, Total length of UDP and Frame Data

④ Checksum : 2bytes

1 - 1 - 4 . Response Frame Data structure

The detailed configuration of the receiving frame data is as follows:

Header	Length	Sync No.	Reserved	Frame type	Data
1 byte	1 byte	1 byte	1 byte (0x00)	1 byte	0 ~ 252 bytes.

① Header : 0xAA, Displays that the beginning of Frame

② Length : Length of Data after Length

(Sync No. + Reserved + Frame type + Data)

③ Reserved : 1 byte (Input as "0x00")

④ Sync No. : The Sync number of the packet is used to check whether the command is executed in the drive module

The value should change every time when you send a new command

⑤ Frame type : Specify the command type of the Frame. The types are listed below

See the section 「Frame type and Data configuration」.

⑥ Data : The data structure and length of this clause are determined by the frame type. The detailed structure is

See the 「Frame typeand Data configuration」 section below.

1 - 1 - 5 . Reply Frame Data Structure and Communication Error

When any command is sent, the basic structure of Frame at the response side is same. However, there is a difference in case of Frame data, which "communication status" is added as shown below.

Header	Length	Sync No.	Reserved	Frame type	Data	
1 byte	1 byte	1 byte	1 byte (0x00)	1 byte	1byte	0 ~ 251 bytes
					Communication status	Response Data

① Header : 0xAA, Displays that the beginning of Frame.

② Length : Length of Data after Length

(Sync No. + Reserved + Frame type + Data)

③ Sync No. : Same as Response ResponseFrame

(If it does not match the data at the time of reception, recognize it as an error

condition.)

④ Reserved : 1 byte(0x00)

⑤ Frame type : Same as Response Frame

(If It does not match the data at the time of transmission, recognize it as an error condition.Sending)

⑥ Data : In reply, 1 byte of data indicating communication status(error/normal) is included.ResponseCommunication status

The simple Execution command has only the communication satus data.Communication status

The contents of byte indicating communication satus are as follows.Communication status

Hexa code	Decimal code	Description
0x00	0	Communication is normal.
0x80	128	Frame type Error : Response Frame type connot be recognized.
0x81	129	Data error, ROM data read/write error. : Data value responded is without the given range.
0x82	130	Received Frame Error : Frame data received is out of this specification.
0xAA	170	CRC Error : A CRC error has occurred due to the influence of ambient noise etc. on the received frame data. In this case, the sending DLL Library automatically tries one more communication.

Caution	**1) If 'Header' and 'Length' calues of the receiving frame are abnormal, there is no response from the drive.Response** **2) If the communication status is displayed to '130', the size of response data is '0' byte.Communication statusResponse**

1 - 2 . Structure of Frame

1 - 2 - 1 . Frame type and Data ConfigurationDescription

(1) The following table displays the content and configuration of data by Frame type.Description

● 0xXX of Frame type is value of Hex, the value in ()is Dec.

<table>
<tr><th>Frame type</th><th>Library Name</th><th>Description</th></tr>
<tr><td>0x01
(1)</td><td>FAS_
GetSlaveInfo</td><td>Connected slave type and program version information are required.

Sending : 0 byte

Reply : 1~248 bytes

<table>
<tr><td>1 byte</td><td>1 byte</td><td>0~246 bytes</td></tr>
<tr><td>Communi
cation
Status</td><td>Slave Type</td><td>ACII string with NULL byte
(strlen() + 1 byte)</td></tr>
</table>
◆ Slave Type
- 150 : Ez-IO-PR-I16 series
- 160 : Ez-IO-PR-O16</td></tr>
<tr><td>0xC0
(192)</td><td>FAS_GetInput</td><td>To read the Input / Latch status

Sending : 0 byte

Response : 9 bytes

<table>
<tr><td>1 byte</td><td>4 bytes</td><td>4 bytes</td></tr>
<tr><td>Communication
Status</td><td>Input status</td><td>Latch status</td></tr>
</table>
Response DATA STRUCTURE
<table>
<tr><td colspan="4">MSB</td><td colspan="4">LSB</td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>63~56</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>55~48</td></tr>
<tr><td>Latch15</td><td>Latch14</td><td>Latch13</td><td>Latch12</td><td>Latch11</td><td>Latch10</td><td>Latch9</td><td>Latch8</td><td>47~40</td></tr>
<tr><td>Latch7</td><td>Latch6</td><td>Latch5</td><td>Latch4</td><td>Latch3</td><td>Latch2</td><td>Latch1</td><td>Latch0</td><td>39~32</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>Input15</td><td>Input14</td><td>Input13</td><td>Input12</td><td>Input11</td><td>Input10</td><td>Input9</td><td>Input8</td><td>15~0</td></tr>
<tr><td>Input7</td><td>input6</td><td>Input5</td><td>Input4</td><td>Input3</td><td>Input2</td><td>Input1</td><td>Input0</td><td>7~0</td></tr>
</table>
0: Off, 1: On</td></tr>
</table>

<table>
<tr>
<td>0xC1
(193)</td>
<td>FAS_ClearLatch</td>
<td>
To Clear (reset) the corresponding bit's latch.

Sending : 4 bytes
<table>
<tr><td>4 bytes</td></tr>
<tr><td>Data</td></tr>
</table>
Sending Data Structure
<table>
<tr><td colspan="4">MSB</td><td colspan="4">LSB</td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>Latch15</td><td>Latch14</td><td>Latch13</td><td>Latch12</td><td>Latch11</td><td>Latch10</td><td>Latch9</td><td>Latch8</td><td>15~0</td></tr>
<tr><td>Latch7</td><td>Latch6</td><td>Latch5</td><td>Latch4</td><td>Latch3</td><td>Latch2</td><td>Latch1</td><td>Latch0</td><td>7~0</td></tr>
</table>
0 : Stat current status
1 : Latch Clear(Reset)

Response : 1byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td>
</tr>
<tr>
<td>0xC2
(194)</td>
<td>FAS_GetLatchCount</td>
<td>
To read Latch Count (0~15)

Sending : 1byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Data</td></tr>
</table>
Data : Latch bit No. (0~15)

Response : 5 bytes
<table>
<tr><td>1 byte</td><td>4 bytes</td></tr>
<tr><td>Communication Status</td><td>Data</td></tr>
</table>
Data : 0~2^{32}
</td>
</tr>
<tr>
<td>0xC3
(195)</td>
<td>FAS_GetLatchCountAll</td>
<td>
To read all of Latch Count 0 to15 (0 ~15)

Sending : 0 byte

Response : 65 bytes
<table>
<tr><td>1 byte</td><td>64 bytes</td></tr>
<tr><td>Communication Status</td><td>Data</td></tr>
</table>
Data Structure
<table>
<tr><td>MSB</td><td></td><td></td><td></td><td></td><td></td><td></td><td>LSB</td></tr>
<tr><td>COUNT15</td><td>COUNT14</td><td>COUNT13</td><td>· · ·</td><td>· · ·</td><td>COUNT2</td><td>COUNT1</td><td>COUNT0</td></tr>
<tr><td>4bytes</td><td>4bytes</td><td>4bytes</td><td></td><td></td><td>4bytes</td><td>4bytes</td><td>4bytes</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>0xC4
(196)</td>
<td>FAS_ClearLatch
Count</td>
<td>
To reset the corresponding bit's Latch Count

Sending : 4bytes
<table>
<tr><td>4 bytes</td></tr>
<tr><td>Data</td></tr>
</table>
Data Structure
<table>
<tr><td colspan="4">MSB</td><td colspan="4">LSB</td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>Latch15</td><td>Latch14</td><td>Latch13</td><td>Latch12</td><td>Latch11</td><td>Latch10</td><td>Latch9</td><td>Latch8</td><td>15~0</td></tr>
<tr><td>Latch7</td><td>Latch6</td><td>Latch5</td><td>Latch4</td><td>Latch3</td><td>Latch2</td><td>Latch1</td><td>Latch0</td><td>7~0</td></tr>
</table>
Response : 1byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td>
</tr>
<tr>
<td>0xC5
(197)</td>
<td>FAS_GetOutput</td>
<td>
To read Output / Trigger(Run/Stop) ststus

Sending : 0byte

Response : 9bytes
<table>
<tr><td>1 byte</td><td>8 bytes</td></tr>
<tr><td>Communication Status</td><td>Data</td></tr>
</table>
Data Structure
<table>
<tr><td colspan="4">MSB</td><td colspan="4">LSB</td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>63~56</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>55~48</td></tr>
<tr><td>Run/Stop15</td><td>Run/Stop14</td><td>Run/Stop13</td><td>Run/Stop12</td><td>Run/Stop11</td><td>Run/Stop10</td><td>Run/Stop9</td><td>Run/Stop8</td><td>47~40</td></tr>
<tr><td>Run/Stop7</td><td>Run/Stop6</td><td>Run/Stop5</td><td>Run/Stop4</td><td>Run/Stop3</td><td>Run/Stop2</td><td>Run/Stop1</td><td>Run/Stop0</td><td>39~32</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>Output15</td><td>Output14</td><td>Output13</td><td>Output12</td><td>Output11</td><td>Output10</td><td>Output9</td><td>Output8</td><td>15~0</td></tr>
<tr><td>Output7</td><td>Output6</td><td>Output5</td><td>Output4</td><td>Output3</td><td>Output2</td><td>Output1</td><td>Output0</td><td>7~0</td></tr>
</table>
0 : OFF (Trigger Stop)
1 : ON (Trigger Run)
</td>
</tr>
</table>

0xC6 (198) — FAS_SetOutput

To set Output (Set : On, Reset : Off)

Sending : 8bytes

4 bytes	4 bytes
Set Output	Reset Output

Data Structure

MSB							LSB	
X	X	X	X	X	X	X	X	63~56
X	X	X	X	X	X	X	X	55~48
Run/Stop15	Run/Stop14	Run/Stop13	Run/Stop12	Run/Stop11	Run/Stop10	Run/Stop9	Run/Stop8	47~40
Run/Stop7	Run/Stop6	Run/Stop5	Run/Stop4	Run/Stop3	Run/Stop2	Run/Stop1	Run/Stop0	39~32
X	X	X	X	X	X	X	X	31~24
X	X	X	X	X	X	X	X	23~16
Output15	Output14	Output13	Output12	Output11	Output10	Output9	Output8	15~0
Output7	Output6	Output5	Output4	Output3	Output2	Output1	Output0	7~0

1 : To execute the Set Output or Reset Out
(Reset is executed when the same bit of Set and Reset is 1)
0 : Do not execute Set Output or Reset Out

Response : 1byte

1 byte
Communication Status

0xC7 (199) — FAS_SetTrigger

To set the Trigger Out

Sending : 13bytes

13 bytes
Data

Data Structure

MSB					LSB
Count	Blank	On time	Blank	Period	Output No.
4bytes	2bytes	2bytes	2bytes	2bytes	1byte

Output No. : Trigger Setup Output NO (0~15)
Period : Trigger output cycle, unit [ms],
Mudt be to input the biger than On time value (2~65,535)
On time : Triger On time output ,Unit [ms],
Mudt be to input the smaller than Period value (1~65,534)
Count : Trigger Output Repeat Count (1~4,294,967,295)

Response : 1 byte

1 byte
Communication Status

<table>
<tr>
<td>0xC8
(200)</td>
<td>FAS_SetRunStop</td>
<td>
To Run or Stop the trigger output of corresponding bit.
To stop even if count is not output when the stop bit is on dutin run

Sending : 13bytes
<table>
<tr><td>13 bytes</td></tr>
<tr><td>Data</td></tr>
</table>
Data Structure
<table>
<tr><td colspan="4">MSB</td><td colspan="4">LSB</td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>63~56</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>55~48</td></tr>
<tr><td>Stop15</td><td>Stop14</td><td>Stop13</td><td>Stop12</td><td>Stop11</td><td>Stop10</td><td>Stop9</td><td>Stop8</td><td>47~40</td></tr>
<tr><td>Stop7</td><td>Stop6</td><td>Stop5</td><td>Stop4</td><td>Stop3</td><td>Stop2</td><td>Stop1</td><td>Stop0</td><td>39~32</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>Run15</td><td>Run14</td><td>Run13</td><td>Run12</td><td>Run11</td><td>Run10</td><td>Run9</td><td>Run8</td><td>15~0</td></tr>
<tr><td>Run7</td><td>Run6</td><td>Run5</td><td>Run4</td><td>Run3</td><td>Run2</td><td>Run1</td><td>Run0</td><td>7~0</td></tr>
</table>
Response : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td>
</tr>
<tr>
<td>0xC9
(201)</td>
<td>FAS_GetTrigger</td>
<td>
To read the current Trigger output count (Output No : 0~15).

Sending : 1byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Data</td></tr>
</table>
Data : Trigger Output bit No (0~15)

Response : 5 bytes
<table>
<tr><td>1 byte</td><td>4 bytes</td></tr>
<tr><td>Communication Status</td><td>Data</td></tr>
</table>
Data : 0~2^{32}
</td>
</tr>
</table>

<table>
<tr>
<td>0xCA
(202)</td>
<td>FAS_GetIOLevel</td>
<td>
To read Input or Output Active Level

Sending : 0 byte

Response : 5 bytes
<table>
<tr><td>1 byte</td><td>4 bytes</td></tr>
<tr><td>Communication Status</td><td>Data</td></tr>
</table>

Response Data Structure
<table>
<tr><td>MSB</td><td></td><td></td><td></td><td></td><td></td><td></td><td>LSB</td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>LEVEL15</td><td>LEVEL14</td><td>LEVEL13</td><td>LEVEL12</td><td>LEVEL11</td><td>LEVEL10</td><td>LEVEL9</td><td>LEVEL8</td><td>15~0</td></tr>
<tr><td>LEVEL7</td><td>LEVEL6</td><td>LEVEL5</td><td>LEVEL4</td><td>LEVEL3</td><td>LEVEL2</td><td>LEVEL1</td><td>LEVEL0</td><td>7~0</td></tr>
</table>
0 : Low Active Level (Latch : Falling edge)
1 : High Active Level (Latch : Rising edge)
</td>
</tr>
<tr>
<td>0xCB
(203)</td>
<td>FAS_SetIOLevel</td>
<td>
To set Input or Output Active Level

Sending : 4 bytes

<table>
<tr><td>4 bytes</td></tr>
<tr><td>Data</td></tr>
</table>
Sending Data Structure
<table>
<tr><td>MSB</td><td></td><td></td><td></td><td></td><td></td><td></td><td>LSB</td><td></td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>31~24</td></tr>
<tr><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>23~16</td></tr>
<tr><td>LEVEL15</td><td>LEVEL14</td><td>LEVEL13</td><td>LEVEL12</td><td>LEVEL11</td><td>LEVEL10</td><td>LEVEL9</td><td>LEVEL8</td><td>15~0</td></tr>
<tr><td>LEVEL7</td><td>LEVEL6</td><td>LEVEL5</td><td>LEVEL4</td><td>LEVEL3</td><td>LEVEL2</td><td>LEVEL1</td><td>LEVEL0</td><td>7~0</td></tr>
</table>
0 : Low Active Level (Latch : Falling edge)
1 : High Active Level (Latch : Rising edge)

Response : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr>
<td>0xCC
(204)</td>
<td>FAS_
LoadIOLevel</td>
<td>To read the ROM memory Active Level
To change the RAM data of the Board to the reading data.

Sending :0 byte

Reply : 1 bytes

<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td>
</tr>
<tr>
<td>0xCD
(205)</td>
<td>FAS_
SaveIOLevel</td>
<td>To save the currently set active level value in ROM memory.. This ensures that the values are saved even when the power is turned off.

Sending : 0 byte

Reply : 1 byte

<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td>
</tr>
</table>

1 - 3 . Type of Program

There are 2 method of programming for Ezi-IO Plus-E.

The first is normally used method that using Visual C++ language under window system of PC. The Library that serviced together with Ezi-SERVOⅡ Plus-E have to be used. Refer to
「2. Library for PC Program」

The second method is to send the command character directly without using the library function. Protocol It is necessary to create a low level protocol program like a test program and it is mainly used when a PLC is used as a host controller.

2 . Library for PC Program (Ver6)

2 - 1 . Library Configuration

To use this library, C++ header file(*.h) and library file(*.lib or *.dll) are required. These files are included in "₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include₩₩ The following contents should be included Description in a source file for development.

#include "₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include₩₩FAS_EziMotionPlusE.h"
#include "₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include₩₩ReturnCodes_Define.h"
#include "₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include₩₩MOTION_DEFINE.h"
#include "₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include₩₩COMM_Define.h"

Also, library files are as follows:

"₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include₩₩EziMotionPlusE.lib"
"₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include₩₩EziMotionPlusE.dll"

A sample program source of using library is included in a "₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩Examples₩₩" folder.

(1) The following table describes values returned when each library(DLL) function is used. The user can check the values returned at the library(DLL) function. In case of low-level programming, this service not provided.

Item	Definition	Returned value	Description
Normal	FMM_OK	0	The function has normally performed the command.
Input Error	FMM_INVALID_SLAVE_NUM	3	Wrong slave number is inputted.
Connection Error	FMC_DISCONNECTED	5	The relevant drive is disconnected.
	FMC_TIMEOUT_ERROR	6	Response delay (100 msec) occurs.
	FMC_CRCFAILED_ERROR	7	Checksum error occurs.
	FMC_RECVPACKET_ERROR	8	Protocol level error occurs in packet that comes from Drive.

(2) The following table shows return values included commonly in all libraries. The user can check the result (communication status, running status) judged by the drive. When the user develops programs by using protocols without libraries (DLL), they are available as well.

Item	Definition	Returned value	Description
Normal	FMP_OK	0	Communication has been normally performed.
Input Error	FMP_FRAMETYPEERROR	128	The drive cannot recognize the command.
	FMP_DATAERROR	129	Input data is out of the range
Connection Error	FMP_PACKETERROR	130	Protocol level error occurs in packet that Drive's received.
	FMP_PACKETCRCERROR	170	CRC value is not correct in packet that Drive's received.

2 - 2 . Board Link Function

Function Name	Description
FAS_Connect	The drive tries to connect communication with the drive module: The drive tries to connect communication with the drive module: When it is successfully connected, TRUE will return. Otherwise, FALSE will return.
FAS_Close	The drive tries to disconnect communication with the drive module.
FAS_GetSlaveInfo	The drive reads drive type and program version: Drive type and version information will return.
FAS_IsSlaveExist	The drive checks whether there is the relevant drive: When it exists, TRUE will return. Otherwise, FALSE will return
FAS_EnableLog	To select the communication error log function ON/OFF : When it exists, TRUE will return. Otherwise, FALSE will return.
FAS_SetLogPath	To set the saved folder name of error log file : When folder exists, TRUE will return. Otherwise, FALSE will return.

FAS_Connect

FAS_Connect is the function of connecting Ezi-IO Plus-E.

Syntax

```
BOOL FAS_Connect(
    BYTE sb1, BYTE sb2, BYTE sb3, BYTE sb4
    BYTE iBdID
);
```

Parameters

sb1~4

Enter the IP address of the drive to connect into.

ex) In case of 192.168.0.2

sb1 = 192, sb2 = 168, sb3=0, sb4=2

iBdID

Unique ID of board to connect. The ID (value) set by the user.

It can not be used the same ID as an IP address.

Return Value

When it is successfully connected, TRUE will returns. Otherwise, FALSE will return.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInit()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0 // unique board number of 192.168.0.2
        char lpBuff[256];
        int nBuffSize = 256;
        BYTE nType;
        int nRtn;

        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed..
                MessageBox(_T("connection fail!"));
                return;
        }
```

```
        if (FAS_IsSlaveExist(iBdID) == FALSE)
        {
                // The board number does not exist.
                // Please check the board number of Ezi-SERVO II.
                return;
        }

        nRtn = FAS_GetSlaveInfo(iBdID, &nType, lpBuff, nBuffSize);
        if (nRtn != FMM_OK)
        {
                // Command has not been performed properly..
                // Refer to ReturnCodes_Define.h.
        }

        printf("₩tType : %d ₩n", nType);
        printf("₩tVersion : %d ₩n", lpBuff);

        // Disconnect.
        FAS_Close(iBdID);
}
```

See Also

FAS_Close

FAS_Close

To disconnect the serial port being used

Syntax

```
void FAS_Close(
    BYTE iBdID
);
```

Parameters

iBdID

Port number to disconnect

Remarks

Example

Refer to 'FAS_Connect' library.

See Also

FAS_Connect

FAS_GetSlaveInfo

To get the version information string of the relevant drive

Syntax

```
int FAS_GetSlaveInfo(
    BYTE iBdID,
    BYTE pType,
    LPSTR lpBuff,
    int nBuffSize
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function.

pType

Relevant board type number.

lpBuff

Buffer pointer to get version information string

nBuffSize

lpBuff memory allocation size

Return Value

FMM_OK : Command has been normally performed.

FMM_NOT_OPEN : Board has not been connected yet.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

Example

Refer to 'FAS_Connect' library.

See Also

FAS_IsSlaveExist

To check that the drive is connected.

Syntax

```
BOOL FAS_IsSlaveExist(
    BYTE iBdID,
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function.

Return Value

TRUE : The drive is connected.
FALSE : The drive is disconnected.

Remarks

This function is provided from the library only and it is inapplicable to the protocol program mode.

Example

Refer to 'FAS_Connect' library.

See Also

FAS_Connect

FAS_EnableLog

Controls output of communication error related log.

Syntax

void FAS_EnableLog(BOOL bEnable);

Parameters

bEnable

Select output of Log.

Remarks

Select the Log output during Ezi-MOITON Plus-R DLL function used. This setup Does not affect the other process or other program.
Log function start from 'FAS_Connect' function, the Log output is ended when the 'FAS_Close' is excuted.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcDisableLog()
{
        // The function of Log is not output. After this

        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0 // unique board number of 192.168.0.2

        // Try to connect.
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed..

                return;
        }

        // Disconnect.
        FAS_Close(iBdID);
}
```

See Also

FAS_SetLogPath

FAS_SetLogPath

Setup the folder path of Log output files.

Syntax

BOOL FAS_SetLogPath(LPCTSTR lpPath);

Parameters

lpPath

Folder path Character string of Log output file.

Return Value

If the folder name does not exist or can not access, return FALSE.

Remarks

This function has to be called before FAS_Connect library.
If the lpPath value is NULL or the length is 0, the Log path is selected to Ezi-MOTION Plus-R Library folder. The default value for Log path is NULL that the current library and program exist folder.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcEnableLog()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0 // unique board number of 192.168.0.2

        // Log output.
        FAS_EnableLog(TRUE);   // Do not need to use.

        if (!FAS_SetLogPath(_T("C:₩₩Logs₩₩"))) // C:₩Logs floder have to be exist
        {
                // Log path does not exist.
                Return;
        }

        // All Log output is saved in C:₩Logs folder.

        // Try to connect.
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed..

                return;
        }

        // Disconnect.
        FAS_Close(iBdID);
}
```

See Also

FAS_EnableLog

2 - 3 . Active Level Related Functions

Function Name	Description
FAS_GetIOLevel	To read the Input or Output Active Level.
FAS_SetIOLevel	To set the Input or Output Active Level.
FAS_ SaveIOLevel	To save the currently set active level value in ROM memory. This ensures that the values are saved even when the power is turned off.
FAS_LoadIOLevel	To road the Input Active Level or Output Active LeveL into RAM area from ROM area

FAS_GetIOLevel

To read the Input or Output Active Level.

Syntax

```
int FAS_GetInputLevel(
    BYTE iBdID,
    unsigned long* uIOLevel
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function.

uIOLevel

The variable pointer to which the Active Level value is to be saved

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_NOT_OPEN : Board has not been connected yet.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcIOStatus()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;      //   Unique number of Board
        unsigned long uInputLevel;

        int nRtn;

        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed..

                return;
        }

        nRtn = FAS_GetInputLevel(m_ iBdID, &uIOLevel);
```

```
        _ASSERT(nRtn == FMM_OK);

        // Disconnect.
        FAS_Close(iBdID);
    }
```

See Also

FAS_SetIOLevel

To set the Input or Output Active Level.

Syntax

```
int FAS_SetIOLevel(
    BYTE iBdID,
    unsigned long uIOLevel
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function.

uIOLevel

The value to set the Active Level

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_NOT_OPEN : Board has not been connected yet.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInputStatus()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;      //  Unique number of Board
        unsigned long uIOLevel = 0x0000FF00; // 0~7: low active,  8~15 : high active

        int nRtn;

        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed..

                return;
        }

        nRtn = FAS_SetIOLevel(iBdID, uIOLevel);
```

```
        _ASSERT(nRtn == FMM_OK);

        // Disconnect.
        FAS_Close(iBdID);
    }
```

See Also

FAS_SaveIOLevel

To save the currently set active level value in ROM memory.

Syntax

```
Int FAS_SaveIOLevel(
    BYTE iBdID,
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function..

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_NOT_OPEN : Board has not been connected yet.

FMM_INVALID_PORT_NUM : The nPort port does not exist among the connected ports

FMM_INVALID_SLAVE_NUM : There is no slave of iSlaveNo in the corresponding port.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcModifyParameter()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;       // Unique number of Board

        int nRtn;

        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed..

                return;
        }
        // To save to ROM
        nRtn = FAS_SavetIOLevel(nPortNo, iSlaveNo)
        _ASSERT(nRtn == FMM_OK); // If the command was not executed normally, it stops.

        FAS_Close(iBdID);
}
```

See Also

FAS_LoadIOLevel

The Input or Output Active Level value saved in the ROM area is recalled.

Syntax

```
int FAS_LoadIOLevel(
    BYTE iBdID,
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function..

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_NOT_OPEN : Board has not been connected yet.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcModifyParameter()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;       //   Unique number of Board

        int nRtn;

        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed..

                return;
        }

         // To Load Input or Output Level values from ROM to RAM
        nRtn = FAS_LoadIOLevel(nPortNo, iSlaveNo)
        _ASSERT(nRtn == FMM_OK); // If the command was not executed normally, it stops.

        FAS_Close(iBdID);
}
```

See Also

2 - 4 . Input Control function

Function Name	Description
FAS_GetInput	To read the Input / Latch ststus
FAS_ClearLatch	To clear the corresponding bit's latch.
FAS_GetLatchCount	To read the corresponding bit's latch.
FAS_GetLatchCountAll	To read all of Latch Count 0 to15 (0~15CH)
FAS_ClearLatchCount	To reset the corresponding bit's Latch Count (0)

FAS_GetInput

To read the Input / Latch ststus

Syntax

```
int FAS_GetInput(
    BYTE iBdID,
    unsigned long* uInput,
    unsigned long* uLatch
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function.

uInput

The variable pointer where the input value is to be saved.

uLatch

The variable pointer where the Latch value is to be saved.

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInputStatus()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;      //   Unique number of Board
        unsigned long uInput;
        unsigned long uLatch;

        int nRtn;

        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed..
```

```
            return;
        }

        nRtn = FAS_GetInput(iBdID, &uInput, &uLatch);

        _ASSERT(nRtn == FMM_OK);

        // Disconnect.
        FAS_Close(iBdID);
    }
```

See Also

FAS_ClearLatch

To clear the corresponding bit's latch.

Syntax

```
int FAS_ClearLatch(
    BYTE iBdID,
    unsigned long uLatchMask
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function.

uLatchMask

The bitmask value to clear the latch.

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_NOT_OPEN : Board has not been connected yet.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInputStatus()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;      //  Unique number of Board
        unsigned long uLatchMask = 0x0000FFFF;  // (Latch 0~15 Clear)

        int nRtn;

        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed..

                return;
        }

        nRtn = FAS_ClearLatch(iBdID, uLatchMask);
```

```
        _ASSERT(nRtn == FMM_OK);

        // Disconnect.
        FAS_Close(iBdID);
    }
```

See Also

FAS_GetLatchCount

To read the corresponding bit's latch.

Syntax

```
int FAS_ClearLatchCount(
    BYTE iBdID,
    unsigned char iInputNo,
    unsigned long* uCount
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function.

iInputNo

Latch number to read the latch count.

uCount

The variable pointer where the count value is to be saved.

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_NOT_OPEN : Board has not been connected yet.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInputStatus()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;       //   Unique number of Board

        unsigned iInputNo =0;   // (Latch count No., 0~15)
        unsigned long* uCount;

        int nRtn;

        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
```

```
            // Connection failed..

            return;
        }

        nRtn = FAS_GetLatchCount(iBdID, iInputNo, &uCount);

        _ASSERT(nRtn == FMM_OK);

        // Disconnect.
        FAS_Close(iBdID);
    }
```

See Also

FAS_GetLatchCountAll

To read all of Latch Count 0 to15 (0~15CH)

Syntax

```
int FAS_GetLatchCountAll(
    BYTE iBdID,
    unsigned long** ppuAllCount
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function..

ppuAllCount

The variable pointer where the latch count value (0 to 15) is to be saved

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_NOT_OPEN : Board has not been connected yet.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInputStatus()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;      //   Unique number of Board

        unsigned long Latch_All_Count[16] = 0 ; // Latch count Variable to be saved ( 0 ~ 16 )

        int nRtn;

        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed..

                return;
        }
```

```
        nRtn = FAS_GetLatchCountAll(iBdID,
                (unsigned long**) Latch_All_Count));

        _ASSERT(nRtn == FMM_OK);

        // Disconnect.
        FAS_Close(iBdID);
    }
```

See Also

FAS_ClearLatchCount

To reset the corresponding bit's Latch Count (0)

Syntax

```
int FAS_ClearLatchCount(
    BYTE iBdID,
    unsigned long uInputMask
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function..

uInputmask

The bitmask value of the latch number to reset (0) the latch count

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_NOT_OPEN : Board has not been connected yet.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInputStatus()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;      //   Unique number of Board

        unsigned long uInputmask = 0x000000FF ;   // To reset the Latch count 0 to 7
        int nRtn;

        // Try to connect.
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed..

                return;
        }
```

```
        nRtn = FAS_ClearLatchCount (iBdID, uInputMask);

        _ASSERT(nRtn == FMM_OK);

        // Disconnect.
        FAS_Close(iBdID);
    }
```

See Also

2 - 5 . Output Control Function

Function Name	Description
FAS_GetOutput	To read the Output / Trigger(Run/Stop) ststus
FAS_SetOutput	To set the Output (Set : On, Reset : Off)
FAS_SetTrigger	To set the Trigger
FAS_SetRunStop	To set the run or stop of Trigger.
FAS_GetTriggerCount	To read the count of selected Trigger number.

FAS_GetOutput

To read the Output / Trigger(Run/Stop) ststus

Syntax

```
int FAS_GetOutput(
    BYTE iBdID,
    unsigned long* uOutput,
    unsigned long* uStatus
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function..

uOutput

The variable pointer where the output value is to be saved.

uStatus

The variable pointer where the Run / Stop value of Trigger to be saved.

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInputStatus()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;       //   Unique number of Board

        unsigned long uOutput;
        unsigned long uStatus;

        int nRtn;

        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed.
```

```
            return;
        }

        nRtn = FAS_GetOutput(iBdID, &uOutput, &uStatus);

        _ASSERT(nRtn == FMM_OK);

        // Disconnect.
        FAS_Close(iBdID);
    }
```

See Also

FAS_SetOutput

To set the Output

Syntax

```
int FAS_SetOutput(
    BYTE iBdID,
    unsigned long uSet,
    unsigned long uClear
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function..

uSet

Bitmask value to turn on the output

uClear

The bitmask value to turn off the output.

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

If uSet and uClear set the same bit, it will be treated as uClear.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInputStatus()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;       //   Unique number of Board

        unsigned long uSet = 0x0000FF00; (Turn On Output No. 8~15.)
        unsigned long uClear = 0x000000FF; ( Turn Off Output No. 0~7.)

        int nRtn;

        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed.
```

```
            return;
        }

        nRtn = FAS_SetOutput(iBdID, uSet, uClear);

        _ASSERT(nRtn == FMM_OK);

        // Disconnect.
        FAS_Close(iBdID);
    }
```

See Also

FAS_SetTrigger

To set the Trigger

Syntax

```
int FAS_SetTrigger(
    BYTE iBdID,
    unsigned char uOutputNo,
    TRIGGER_INFO* pTrigger
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function.

uOutputNo

Output number to set the Trigger

The structure variable to set Trigger (Count, OnTime, Period)

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_NOT_OPEN : Board has not been connected yet.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

The unit of Ontime is [ms]

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInputStatus()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;      //   Unique number of Board

        TRIGGER_INFO Trg_info;
        unsigned char uOutputNo = 0;

        int nRtn;
        Trg_info.wCount   = 100; // To set the Trigger output count
        Trg_info.wOnTime   = 10; // To set the On time
        Trg_info.wPeriod   = 30 ; // To set period in every 30[ms], On : 10[ms] Off : 20[ms]
        Trg_info.wReserved1 = 0;
        Trg_info.wReserved2 = 0;
```

```
        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed..
                return;
        }

        nRtn = FAS_SetTrigger(iBdID, uOutputNo, &Trg_info);

        _ASSERT(nRtn == FMM_OK);

        // Disconnect.
        FAS_Close(iBdID);
}
```

See Also

FAS_SetRunStop

To set the run or stop of Trigger.

Syntax

```
int FAS_SetRunStop(
    BYTE iBdID,
    unsigned long uRun,
    unsigned long uStop
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function.

uRun

The bitmask value to excute the Trigger

uStop

The bitmask value to stop the Trigger

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

When uRun and uStop set the same bit, it will be treated as uStop.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInputStatus()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;       //   Unique number of Board

        unsigned long uRun = 0x0000FF00; ( to execute the Trigger 8~15 .)
        unsigned long uStop = 0x000000FF; ( to excute the Trigger 0~7.)


        int nRtn;

        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
```

```
                // Connection failed..

                return;
        }

        nRtn = FAS_SetRunStop(iBdID, uRun, uStop);

        _ASSERT(nRtn == FMM_OK);

        // Disconnect.
        FAS_Close(iBdID);
}
```

See Also

FAS_GetTriggerCount

To read the count of selected Trigger number.

Syntax

```
int FAS_GetTriggerCount(
    BYTE iBdID,
    unsigned char uOutputNo,
    unsigned long* uCount
);
```

Parameters

iBdID

The ID number of the board. The iBdID set by the FAS_Connect function.

uOutputNo

Output number which to read the trigger count.

uCount

The variable pointer where to be saved the value of Trigger Count

Return Value

FMM_OK : COMMAND HAS BEEN NORMALLY PERFORMED.

FMM_INVALID_SLAVE_NUM : The Board of the corresponding iBdID does not exist.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInputStatus()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        BYTE iBdID = 0;       //   Unique number of Board

        unsigned char uOutputNo = 1; (Output No. 0~15)
        unsigned long* uCount;

        int nRtn;

        // Try to connect .
        if (FAS_Connect(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed
```

```
            return;
        }

        nRtn = FAS_GetTriggerCount(iBdID, uOutputNo, &uCount);

        _ASSERT(nRtn == FMM_OK);

        // Disconnect.
        FAS_Close(iBdID);
    }
```

See Also

FASTECH

FASTECH Co., Ltd.

Rm #1202, Bucheon Technopark 401 Dong, Yakdae-dong,
Wonmi-Gu, Bucheon-si, Gyeonggi-do, Rep. Of Korea(Zip:420-734)
TEL : 82-32-234-6300, 6301 FAX : 82-32-234-6302
Email : fastech@fastech.co.kr Homepage : www.fastech.co.kr