Ezi-SERVO® II Plus-E ALL

Closed Loop Stepping System

User Manual

Communication Function

(**Rev.03**)

FASTECH

Table of Contents

Table of Contents 2
1. Communication Protocol 3
1－1. Communication Functions 3
1－1－1. Communication Specifications 3
1－1－2. Ethernet IP address 3
1－1－3. Ethernet Protocol 3
1－1－4. Resopnse Frame Data Structure 4
1－1－5. Reply Frame Structure and Communication Error 4
1－2. Structure of Frame 6
1－2－1. Frame type and Data Configuration 6
1－2－2. Parameter Lists 23
1－2－3. Bit setup of Output pin 25
1－2－4. Bit setup of Input pin 26
1－2－5. Bit setup of Status Flag 27
1－2－6. Position Table Item 28
1－3. Program Method 30
2. Library for PC Program 31
2－1. Library Configuration 31
2－2. Communication status Window 34
2－3. Drive Connection Function 37
2－4. Parameter Control Function 55
2－5. Servo Control Function 61
2－6. Control I/O Function 65
2－7. Position Control Function 77
2－8. Drive Status Control Function 88
2－9. Operation Control Function 94
2－10. Position Table Control Function 119
2－11. Other Control Function 128
3. Appendix – Configuring Network Information with DHCP 134
3－1. DHCP function 134
3－2. Network configuration using DHCP(Plus-E series) 134

1. Communication Protocol

1 - 1 . Communication Functions

Ezi-SERVOⅡ Plus-E ALL can control up to 254(1~254) axes by multidrop link at Ethernet

1 - 1 - 1 . Communication Specifications

Item	Specifications
Communication Speed	10/100base-T/TX
Communication Type(Protocol)	TCP/ (Port No. : **2001,2002**)
	UDP (Port No. : **3001,3002**)
Max Cabling Length	100m
Min Cable length between drive	More than 20cm
Number of Connected Axes	254 axes (No. 01~FE)

- Port No. 2001, 3001 : For GUI
- Port No. 2002, 3002 : For User Library
- When using the user Library file, you cannot use Port No. 2001, 3001.

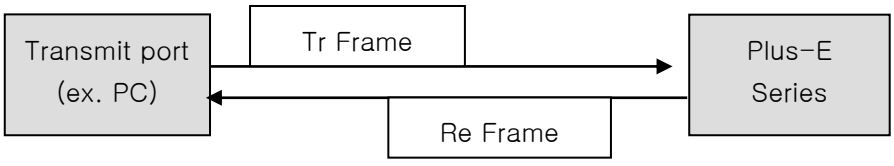
1 - 1 - 2 . Ethernet IP address

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

- When connecting to Ezi-SERVOⅡ Plus-E ALL 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*
8 bytes	4~254 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 ~ 253 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. Refer to the 「Frame typeand Data configuration」 section below for more detailed information.

1 - 1 - 5 . Reply Frame 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.

<table>
<tr><th>Header</th><th>Length</th><th>Sync No.</th><th>Reserved</th><th>Frame type</th><th colspan="2">Data</th></tr>
<tr><td rowspan="2">1 byte</td><td rowspan="2">1 byte</td><td rowspan="2">1 byte</td><td rowspan="2">1 byte (0x00)</td><td rowspan="2">1 byte</td><td>1 byte</td><td>0 ~ 252 bytes</td></tr>
<tr><td>Communication status</td><td>Response Data</td></tr>
</table>

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 Frame
(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.)

⑥ Data : In reply, 1 byte of data indicating communication status(error/normal) is included.
The simple Execution command has only the communication satus data.

The contents of byte indicating communication satus are as follows.

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.
0x85	133	Running Command Failure: The user has tried to execute new running commands in wrong condition as follows. 1) Currently motor is running 2) Currently motor is stopping 3) Currently Servo is OFF status 4) Try to Z-pulse Origin without encoder 5) Other wrong motion command
0x86	134	RESET Failure : The user has tried to execute new running commands in wrong condition as follows. 1) Already RESET in ON by external input signal
0x87	135	Servo ON failed ①: Attempt to execute Servo ON command while an alarm occurred.
0x88	136	Servo ON failed ②: Attempt to execute Servo ON command during emergency stop.
0x89	137	Servo ON failed ③: 'Servo ON' is set for external input signal. Servo ON / OFF can be executed only with this input signal.

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

1 - 2 . Structure of Frame

1 - 2 - 1 . Frame type and Data Configuration

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

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

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

Sending : 0 byte
Response : 1~248 bytes

<table>
<tr><td>1 byte</td><td>1 byte</td><td>0~253 bytes</td></tr>
<tr><td>Communication Status</td><td>Board type</td><td>ACII string with NULL byte
(strlen() + 1 byes)</td></tr>
</table>
◆ board : 91 (Ezi-SERVOⅡ Plus-E ALL)</td></tr>
<tr><td>0x05
(5)</td><td>FAS_
GetMotorInfo</td><td>Connected motor type and maker information are required.

Sending : 0 byte
Response : 1~246 bytes

<table>
<tr><td>1 byte</td><td>1 byte</td><td>0~246 bytes</td></tr>
<tr><td>Communication Status</td><td>Motor Number
(1~255)</td><td>ACII string with NULL byte
(strlen() + 1 byes)</td></tr>
</table>
◆ Motor Type : Refer to 「1-2-7. Motor Type」.</td></tr>
<tr><td>0x10
(16)</td><td>FAS_
SaveAllParameters</td><td>Current setting parameters & assign of IO signals are saved in the ROM of the drive. Even though the drive is powered off, saving these must be possible. Values set at ‘FAS_SetParameter’ &‘FAS_SetIOAssignMap’ are saved together.

Sending : 0 byte
Response : 1 byte

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

<table>
<tr>
<td>0x11
(17)</td>
<td>FAS_
GetRomParameter</td>
<td>Specific parameter values in the ROM are read.

Sending : 1 byte

<table>
<tr><th>1 byte</th></tr>
<tr><td>Parameter No. (0~32)</td></tr>
</table>

Response : 5 bytes

<table>
<tr><th>1 byte</th><th>4 bytes</th></tr>
<tr><td>Communication Status</td><td>Parameter value</td></tr>
</table>

Refer to「1-2-2. Parameter List」</td>
</tr>
<tr>
<td>0x12
(18)</td>
<td>FAS_
SetParameter</td>
<td>Specific parameter values are saved to the RAM.

Sending : 5 bytes

<table>
<tr><th>1 byte</th><th>4 bytes</th></tr>
<tr><td>Parameter No. (0~32)</td><td>Parameter Value</td></tr>
</table>

Response : 1 byte

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

Refer to「1-2-2. Parameter List 」</td>
</tr>
<tr>
<td>0x13
(19)</td>
<td>FAS_
GetParameter</td>
<td>Specific parameter values in the RAM are read.

Send : 1 byte

<table>
<tr><th>1 byte</th></tr>
<tr><td>Parameter No. (0~32)</td></tr>
</table>

Response : 5 bytes

<table>
<tr><th>1 byte</th><th>4 bytes</th></tr>
<tr><td>Communication Status</td><td>Parameter Value</td></tr>
</table>

Refer to「1-2-2. Parameter List 」</td>
</tr>
</table>

<table>
<tr>
<td>0x20
(32)</td>
<td>FAS_
SetIOOutput</td>
<td>Output signal level of the control output port is set.

Sending : 8 bytes
<table>
<tr><td>4 bytes</td><td>4 bytes</td></tr>
<tr><td>I/O set mask value</td><td>I/O clear mask value</td></tr>
</table>
When specific bit of the set mask is '1', the relevant output port signal is set to [ON].
When specific bit of the clear mask is '1', the relevant output port signal is set to [OFF].
For more information, refer to 「1-2-3. Bit setup of Output Pin」.

Response : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td>
</tr>
<tr>
<td>0x21
(33)</td>
<td>FAS_
SetIOInput</td>
<td>Input signal level of the control input port is set.
Sending : 8 bytes
<table>
<tr><td>4 bytes</td><td>4 bytes</td></tr>
<tr><td>I/O set mask value</td><td>I/O clear mask value</td></tr>
</table>
When specific bit of the set mask is '1', the relevant input port signal is set to [ON].
When specific bit of the clear mask is '1', the relevant input port signal is set to [OFF].
For more information, refer to 「1-2-4. Bit setup of Input Pin」.

Response : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td>
</tr>
<tr>
<td>0x22
(34)</td>
<td>FAS_
GetIOInput</td>
<td>Current output signal status of the control output port is read.

Sending : 0 byte

Response : 5 bytes
<table>
<tr><td>1 byte</td><td>4 bytes</td></tr>
<tr><td>Communication Status</td><td>Input status value</td></tr>
</table>
For relevant bit by each output signal, refer to 「1-2-4. Bit setup of Input Pin」.</td>
</tr>
<tr>
<td>0x23
(35)</td>
<td>FAS_
GetIOOutput</td>
<td>Current output signal status of the control output port is read.

Sending : 0 byte</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td>
Response : 5 bytes
<table>
<tr><td>1 byte</td><td>4 bytes</td></tr>
<tr><td>Communication Status</td><td>Output status value</td></tr>
</table>
For relevant bit by each output signal, refer to 「1-2-3. Bit setup of Output Pin」.
</td></tr>
<tr><td>0x24
(36)</td><td>FAS_
SetIOAssignMap</td><td>
To assign control I/O signals to the pin of CN1 port and set the signal level. By running 'FAS_SaveAllParameters', you can save the setting value to the ROM.

Sending : 6 bytes
<table>
<tr><td>1 byte</td><td>4 bytes</td><td>1 byte</td></tr>
<tr><td>I/O number</td><td>I/O pin masking data</td><td>Setting level</td></tr>
</table>
◆I/O number : '0~5' corresponds to 'Limit+, Limit-, Org,IN1,..., IN3' respectively, and '6~7' corresponds to 'COMP, OUT1' respectively.
◆I/O pin masking data : Refer to 「1-2-4. Bit setup of Input Pin」.
◆Level Setting: 0:Active Low, 1:Active High
Response : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td></tr>
<tr><td>0x25
(37)</td><td>FAS_
GetIOAssignMap</td><td>
Pin setting status of CN1 port is read from RAM area.

Sending : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>I/O number</td></tr>
</table>
◆I/O number : '0~5' corresponds to 'Limit+, Limit-, Org, IN1, ..., IN3' respectively, and '6~7' corresponds to 'COMP, OUT1' respectively.

Response : 6 bytes
<table>
<tr><td>1 byte</td><td>4 bytes</td><td>1 byte</td></tr>
<tr><td>Communication Status</td><td>IO pin masking status</td><td>Level status</td></tr>
</table>
For more information, refer to '0x24' Frame type.
</td></tr>
<tr><td>0x26
(38)</td><td>FAS_
IOAssignMapReadROM</td><td>
Pin setting status of CN1 port is loaded to RAM from ROM area.

Sending : 0 byte
Response : 2 bytes
<table>
<tr><td>1 byte</td><td>1 byte</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>Communication Status</td><td>Command performing status
(0 : complete, values except 0: error)</td></tr>
</table>
</td></tr>
<tr><td>0x27
(39)</td><td>FAS_
TriggerOutput_Run
A</td><td>
Start/Stop command for 'Compare Out' signal.

Sending : 18 bytes
<table>
<tr><td>1 byte</td><td>4 bytes</td><td>4 bytes</td></tr>
<tr><td>Output start/stop
(1:start 0:stop)</td><td>Pulse start position
[pulse]</td><td>Pulse period
[pulse]</td></tr>
</table>
<table>
<tr><td>4 bytes</td><td>1 byte</td><td>4 bytes</td></tr>
<tr><td>Pulse width
[msec]</td><td>Output pin number
(fixed to 0)</td><td>spare</td></tr>
</table>
♦ Pulse start position: Setting the start position of first pulse output.
(-134,217,727 ~134,217,727)
♦ Pulse period: Setting the pulse period. (1 ~134,217,727)
(0: pulse output only 1 time in pulse start position
1~ : pulse output repeatedly depends on setting)
♦ Pulse width: Setting the pulse width. (1~1000)

Response : 2 bytes
<table>
<tr><td>1 byte</td><td>1 byte</td></tr>
<tr><td>Communication Status</td><td>Command performing status
(0 : complete, values except 0: error)</td></tr>
</table>
● For pulse period, input the value that is over 2 [ms] by adding the pulse width. If it is less than that, it will not work properly.
</td></tr>
<tr><td>0x28
(40)</td><td>FAS_
TriggerOutput_
Status</td><td>
Command to check if the trigger output pulse(Compare out) is working or not.

Sending : 0 byte
Response : 2 bytes
<table>
<tr><td>1 byte</td><td>1 byte</td></tr>
<tr><td>Communication Status</td><td>Status (1:output ON, 0 :output OFF)</td></tr>
</table>
</td></tr>
</table>

<table>
<tr>
<td>0x7E
(126)</td>
<td>FAS_SetTriggerOutputEx</td>
<td>
Setting for generating output at a specific position on the set output
(Available after setting the output signal to User Out)
Sending : 245 bytes
<table>
<tr><th>1 byte</th><th>1 byte</th><th>2 bytes</th><th>1byte</th></tr>
<tr><td>User OutNumber
(0~8)</td><td>Output start/End command
(1: Start 0: End)</td><td>Output On Time
(In ms, 1~65,535)</td><td>Output position count</td></tr>
</table>
<table>
<tr><th>240 bytes</th></tr>
<tr><td>Output position Array(4bytes * 60)
Location: -134,217,728~134,217,727</td></tr>
</table>
♦ Number of ouput position : 1~60
♦ Output position With Array : Based on 4byte, 60 arrays
Even if the number of output positions is not 60, the output position array has 60 information inputs.
♦ Output is automatically close when the number of outputs is reached.
♦ To output, execute the move command after setting.
The position of the move command should be greater than the last position if the last position of the output position is positive and less than the last position if the position is negative
Normal output may not be generated depending on the starting position (current position), so it is necessary to set the output position to an appropriate value.
Depending on the drive speed and output On time setting, normal output does not occur. Therefore, it is necessary to set the proper value.

Response : 1 byte
<table>
<tr><th>1 byte</th></tr>
<tr><td>Communication Status</td></tr>
</table>
</td>
</tr>
<tr>
<td>0x7F
(127)</td>
<td>FAS_GetTriggerOutputEx</td>
<td>
Command to check information and output status set by FAS_SetTriggerOutputEx

Sending : 1byte
<table>
<tr><th>1 byte</th></tr>
<tr><td>User Out Number</td></tr>
</table>
♦ User Out number : User Out number to check the information(0~8)

Response : 245bytes
<table>
<tr><th>1 byte</th><th>1 byte</th><th>2 bytes</th><th>1byte</th></tr>
<tr><td>Communic</td><td>Output</td><td>Output On time</td><td>Number of output</td></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>ation
Status</td><td>status</td><td>(ms 단위, 1~65535)</td><td>location
(1~60)</td></tr>
</table>
<table>
<tr><td>240 bytes</td></tr>
<tr><td>Output location Array(4bytes * 60)
Location: -134,217,728~134,217,727</td></tr>
</table>
◆ Output status : Run/Stop status of corresponding user out number
0 : Stop
2 : Run
</td></tr>
<tr><td>0x2A
(42)</td><td>FAS_
ServoEnable</td><td>
Servo ON/OFF(Enable/Disable) is set
Sending : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>0:OFF, 1:ON</td></tr>
</table>
Response : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td></tr>
<tr><td>0x2B
(43)</td><td>FAS_
ServoAlarmReset</td><td>
Servo Alarm status is reset.
Sending : 0 byte
Response : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td></tr>
<tr><td>0x2E
(46)</td><td>FAS_
GetAlarmType</td><td>
Request current Alarm type and information.

Sending : 0 byte

Response : 2 bytes
<table>
<tr><td>1 byte</td><td>1 byte</td></tr>
<tr><td>Communication Status</td><td>Alarm type (0: No alarm, except 0: Alarm number)</td></tr>
</table>
◆ Alarm type: No alarm (0) OverCurrent(1) OverSpeed(2)
PosTracking(3) OverLoad(4) OverTemperature(5)
BackEMF(6) MotorConnect(7) EncoderConnect(8)
Inposition(10) ROMdevice(12) Position Overflow(15)
</td></tr>
<tr><td>0x31
(49)</td><td>FAS_
MoveStop</td><td>
Request the stop of the motor currently operating.

Sending : 0 byte
Response : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td></tr>
</table>

<table>
<tr><td>0x32
(50)</td><td>FAS_
EmergencyStop</td><td>Request the emergency stop of the motor currently operating.

Sending : 0 byte
Response : 1 byte

<table><tr><td>1 byte</td></tr><tr><td>Communication Status</td></tr></table></td></tr>
<tr><td>0x33
(51)</td><td>FAS_
MoveOriginSingle
Axis</td><td>Request the homing sequence with the current parameter condition.

Sending : 0 byte
Response : 1 byte

<table><tr><td>1 byte</td></tr><tr><td>Communication Status</td></tr></table></td></tr>
<tr><td>0x34
(52)</td><td>FAS_
MoveSingleAxisAbs
Pos</td><td>Request the movement with absolute[pulse] position.

Sending : 8 bytes

<table><tr><td>4 bytes</td><td>4 bytes</td></tr><tr><td>Absolute
position value</td><td>Motor
speed[pps]</td></tr></table>Response : 1 byte

<table><tr><td>1 byte</td></tr><tr><td>Communication Status</td></tr></table></td></tr>
<tr><td>0x35
(53)</td><td>FAS_
MoveSingle
AxisIncPos</td><td>Request the movement with increamental[pulse] position.

Sending : 8 bytes

<table><tr><td>4 bytes</td><td>4 bytes</td></tr><tr><td>Increamental
position</td><td>Motor
speed[pps]</td></tr></table>Response : 1 byte

<table><tr><td>1 byte</td></tr><tr><td>Communication Status</td></tr></table></td></tr>
<tr><td>0x36
(54)</td><td>FAS_
MoveToLimit</td><td>Request the motor to start limit motion at the current setting parameter condition

Sending : 5 bytes

<table><tr><td>4 bytes</td><td>1 byte</td></tr><tr><td>Running
speed[pps]</td><td>Running direction (0:-Limit
1:+Limit)</td></tr></table>Response : 1 byte</td></tr>
</table>

<table>
<tr><td></td><td></td><td><table><tr><td>1 byte</td></tr><tr><td>Communication Status</td></tr></table></td></tr>
<tr><td>0x37
(55)</td><td>FAS_
MoveVelocity</td><td>Request the motor to start jog motion at the current setting parameter condition

Sending : 5 bytes
<table><tr><td>4 bytes</td><td>1 byte</td></tr><tr><td>Running speed[pps]</td><td>Running direction (0:-Jog 1:+Jog)</td></tr></table>Response : 1 byte
<table><tr><td>1 byte</td></tr><tr><td>Communication Status</td></tr></table></td></tr>
<tr><td>0x38
(56)</td><td>FAS_
PositionAbsOverrid
e</td><td>To request the motor to change the target absolute position value[pulse] while it is in running.

Sending : 4 bytes
<table><tr><td>4 bytes</td></tr><tr><td>Changed command position value [pulse]</td></tr></table>Response : 1 byte
<table><tr><td>1 byte</td></tr><tr><td>Communication Status</td></tr></table>◆ Only available at constant velocity section</td></tr>
<tr><td>0x39
(57)</td><td>FAS_
PositionIncOverride</td><td>To request the motor to change the target incremental position value[pulse] while it is in running.
Sending : 4 bytes
<table><tr><td>4 bytes</td></tr><tr><td>Changed command position value[pulse]</td></tr></table>Response : 1 byte
<table><tr><td>1 byte</td></tr><tr><td>Communication Status</td></tr></table>◆ Only available at constant velocity section</td></tr>
</table>

<table>
<tr>
<td>0x3A
(58)</td>
<td>FAS_
VelocityOverride</td>
<td>
Request the motor to change the running speed value[pps] while it is in running.

Sending : 4 bytes
<table>
<tr><td>4 bytes</td></tr>
<tr><td>Changed Running speed[pps]</td></tr>
</table>

The accel/decel time is assigned to 'Axis Acc Time' and 'Axis Dec Time' value in parameter lists.

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

◆ Only available at constant velocity section
</td>
</tr>
<tr>
<td>0x80
(128)</td>
<td>FAS_
MoveSingleAxisAbs
PosEx</td>
<td>
Request to start moving operation by the absolute value [pulse] position including acceleration / deceleration value [msec].

Sending : 40 bytes
<table>
<tr><td>4 bytes</td><td>4 bytes</td><td>4 bytes</td><td>2 bytes</td></tr>
<tr><td>Absolute position value</td><td>Running speed[pps]</td><td>Flag option</td><td>Custom Accel. Time (1~9999)</td></tr>
</table>

<table>
<tr><td>2 bytes</td><td>24 bytes</td></tr>
<tr><td>Custom Decel. Time (1~9999)</td><td>Reserved</td></tr>
</table>

Flag ooption :
0x0001 : reserved
0x0002 : Custom Accel. Time applied.
0x0004 : Custom Decel. Time applied.

If the bit value is OFF (0), the value specified inside the controller is applied.

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

<table>
<tr>
<td>0x81
(129)</td>
<td>FAS_
MoveSingle
AxisIncPosEx</td>
<td>
Request to start moving operation by the incremental value [pulse] position including acceleration / deceleration value [msec].

Sending : 40 bytes
<table>
<tr><td>4 bytes</td><td>4 bytes</td><td>4 bytes</td><td>2 bytes</td></tr>
<tr><td>incremental position value</td><td>Running speed[pps]</td><td>Flag option</td><td>Custom Accel. Time (1~9999)</td></tr>
</table>
<table>
<tr><td>2 bytes</td><td>24 bytes</td></tr>
<tr><td>Custom Decel. Time (1~9999)</td><td>Reserved</td></tr>
</table>
Flag option : 0x0001 : reserved
0x0002 : Custom Accel. Time is applied.
0x0004 : Custom Decel. Time is applied.
If the bit value is OFF (0), the value specified inside the controller is applied.

Response : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td>
</tr>
<tr>
<td>0x82
(130)</td>
<td>FAS_
MoveVelocityEx</td>
<td>
Request the motor to start jog motion at the current setting parameter condition with custom Accel/Decel time value[msec].

Sending : 37 bytes
<table>
<tr><td>4 bytes</td><td>1 byte</td><td>4 bytes</td></tr>
<tr><td>Running speed[pps]</td><td>Running direction (0:-Jog 1:+Jog)</td><td>Flag option</td></tr>
</table>
<table>
<tr><td>2 bytes</td><td>26 bytes</td></tr>
<tr><td>Custom Accel./Decel. Time (1~9999)</td><td>Reserved</td></tr>
</table>
Flag option : 0x0001 : reserved
0x0002 : Apply Custom Accel./Decel. Time value.

If the bit value is OFF (0), the value specified inside the controller is applied.

Response : 1 byte
</td>
</tr>
<tr>
<td>0x40
(64)</td>
<td>FAS_
GetAxisStatus</td>
<td>
Request Flag value to display the running status.

Sending : 0 byte
Response : 5 bytes
<table>
<tr><td>1 byte</td><td>4 bytes</td></tr>
<tr><td>Communication Status</td><td>Status Flag value</td></tr>
</table>
For bit related to each Flag, refer to 「1-2-5. Bit setup of Status Flag」.
</td>
</tr>
</table>

<table>
<tr>
<td>0x41
(65)</td>
<td>FAS_
GetIOAxisStatus</td>
<td>Request the I/O status and the running Flag status.
(Frame type 0x22, 0x23, and 0x40 are packed.)

Sending : 0 byte
Response : 13 bytes

<table>
<tr><th>1 byte</th><th>4 bytes</th><th>4 bytes</th><th>4 bytes</th></tr>
<tr><td>Communication Status</td><td>Input status value</td><td>Output status value</td><td>Status Flag value</td></tr>
</table>
</td>
</tr>
<tr>
<td>0x42
(66)</td>
<td>FAS_
GetMotionStatus</td>
<td>Request the current running progress status and its PT number
(Frame type 0x51, 0x53, 0x54, and 0x55 are packed.)

Sending : 0 byte
Response : 21 bytes

<table>
<tr><th>1 byte</th><th>4 bytes</th><th>4 bytes</th><th>4 bytes</th><th>4 bytes</th><th>4 bytes</th></tr>
<tr><td>Communication Status</td><td>Command Position value</td><td>Actual Position value</td><td>Position difference value</td><td>Motor speed value</td><td>PT No. currently running</td></tr>
</table>
</td>
</tr>
<tr>
<td>0x43
(67)</td>
<td>FAS_
GetAllStatus</td>
<td>Request all data including the current operation status. (Frame type 0x41, 0x42 are bundled)

Sending : 0 byte
Response : 33 bytes
<table>
<tr><th>1 byte</th><th>4 bytes</th><th>4 bytes</th><th>4 bytes</th></tr>
<tr><td>Communication Status</td><td>Input status value</td><td>Output status value</td><td>Status Flag value</td></tr>
</table>
<table>
<tr><th>4 bytes</th><th>4 bytes</th><th>4 bytes</th><th>4 bytes</th><th>4 bytes</th></tr>
<tr><td>Command Position value</td><td>Actual Position value</td><td>Position Difference value</td><td>Operation Speed value</td><td>Currently operating PT No.</td></tr>
</table>
</td>
</tr>
<tr>
<td>0x50
(80)</td>
<td>FAS_
SetCommandPos</td>
<td>If the command position is set before the start of operation, the change of the target position value can be checked later.

Sending : 4 bytes
<table>
<tr><th>4 bytes</th></tr>
<tr><td>Command position setting count value</td></tr>
</table>
Response : 1 byte
<table>
<tr><th>1 byte</th></tr>
</table>
</td>
</tr>
</table>

<table>
<tr><td></td><td></td><td><table><tr><td>Communication Status</td></tr></table></td></tr>
<tr><td>0x51
(81)</td><td>FAS_
GetCommandPos</td><td>Request the command position value[pulse] being tracked.

Sending : 0 byte
Response : 5 bytes
<table><tr><td>1 byte</td><td>4 bytes</td></tr><tr><td>Communication Status</td><td>Command position value</td></tr></table></td></tr>
<tr><td>0x52
(82)</td><td>FAS_
SetActualPos</td><td>Ezi-SERVOⅡ Plus-E ALL is a closed loop stepping system, so the actual position value is continuously updated during operation. If this actual position value is set to this value before starting operation, the change of the actual position value can be checked later.

Sending : 4 bytes
<table><tr><td>4 bytes</td></tr><tr><td>Actual position count value</td></tr></table>
Response : 1 byte
<table><tr><td>1 byte</td></tr><tr><td>Communication Status</td></tr></table></td></tr>
<tr><td>0x53
(83)</td><td>FAS_
GetActualPos</td><td>Request the current Actual position value[pulse].

Sending : 0 byte

Response : 5 bytes
<table><tr><td>1 byte</td><td>4 bytes</td></tr><tr><td>Communication Status</td><td>Actual position value</td></tr></table></td></tr>
<tr><td>0x54
(84)</td><td>FAS_
GetPosError</td><td>Request the difference[pulse] between the command position value and the actual position value.

Sending : 0 byte

Response : 5 byte
<table><tr><td>1 byte</td><td>4 bytes</td></tr><tr><td>Communication Status</td><td>Position difference value</td></tr></table>
With this value, you can check the current operation status (how inposition is getting tracked).</td></tr>
</table>

<table>
<tr>
<td>0x55
(85)</td>
<td>FAS_
GetActualVel</td>
<td>To request the current running speed value [pps]

Sending : 0 byte
Response : 5 bytes

<table>
<tr><td>1 byte</td><td>4 bytes</td></tr>
<tr><td>Communication Status</td><td>Speed value</td></tr>
</table>
</td>
</tr>
<tr>
<td>0x56
(86)</td>
<td>FAS_
ClearPosition</td>
<td>Set both the command position value and actual position value to ‘0’.

Sending : 0 byte
Response : 1 byte

<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td>
</tr>
<tr>
<td>0x58
(88)</td>
<td>FAS_
MovePause</td>
<td>Request pause and release pause of the current operation.

Sending : 1 byte

<table>
<tr><td>1 byte</td></tr>
<tr><td>0: Release pause, 1: Pause</td></tr>
</table>
Response : 1 byte

<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td>
</tr>
<tr>
<td>0x60
(96)</td>
<td>FAS_
PosTableReadItem</td>
<td>Read PT values in the RAM of the drive.

Sending : 2 bytes

<table>
<tr><td>2 bytes</td></tr>
<tr><td>PT No. to be read(0~255)</td></tr>
</table>
Response : 65 bytes

<table>
<tr><td>1 byte</td><td>64 bytes</td></tr>
<tr><td>Communication Status</td><td>Relevant PT values</td></tr>
</table>
For items by each PT, refer to 「1-2-6. Position Table Item」.</td>
</tr>
<tr>
<td>0x61
(97)</td>
<td>FAS_
PosTableWriteItem</td>
<td>Save PT values to the RAM of the drive.

Sending : 66 bytes

<table>
<tr><td>2 bytes</td><td>64 bytes</td></tr>
<tr><td>PT No. to be read(0~255)</td><td>Relevant PT values</td></tr>
</table>
For items by each PT, refer to 「1-2-6. Position Table Item」.

Response : 2 bytes

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

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>Communication Status</td><td>Command performance (values except 0 : complete, 0: error)</td></tr>
</table>
</td></tr>
<tr><td>0x62
(98)</td><td>FAS_
PosTableReadROM</td><td>Read all PT values (256) in the ROM of the drive

Sending : 0 byte

Response : 2 bytes
<table>
<tr><td>1 byte</td><td>1 byte</td></tr>
<tr><td>Communication Status</td><td>Command performance (values except 0 : complete, 0: error)</td></tr>
</table>
</td></tr>
<tr><td>0x63
(99)</td><td>FAS_
PosTableWriteROM</td><td>Save all PT values (256) in the ROM of the drive

Sending : 0 byte

Response : 2 bytes
<table>
<tr><td>1 byte</td><td>1 byte</td></tr>
<tr><td>Communication Status</td><td>Command performance (values except 0 : complete, 0: error)</td></tr>
</table>
</td></tr>
<tr><td>0x64
(100)</td><td>FAS_
PosTableRunItem</td><td>Start the position table operation from the designated PT number

Sending : 2 bytes
<table>
<tr><td>2 bytes</td></tr>
<tr><td>PT No.(0~255)</td></tr>
</table>
Response : 1 byte
<table>
<tr><td>1 byte</td></tr>
<tr><td>Communication Status</td></tr>
</table>
</td></tr>
<tr><td>0x6A
(106)</td><td>FAS_
PosTableReadOneItem</td><td>Read the specific PT values in the RAM of the drive.

Sending : 4 bytes
<table>
<tr><td>2 bytes</td><td>2 bytes</td></tr>
<tr><td>PT No. to be read(0~255)</td><td>Offset value(0~40) of the specific items to be read</td></tr>
</table>
Refer to 「1-2-6. Position Table Item」 for offset value.

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

<table>
<tr><td></td><td></td><td>
<table>
<tr><td>Communication Status</td><td>The value of the specific item</td></tr>
</table>
</td></tr>
<tr><td>0x6B
(107)</td><td>FAS_
PosTableWriteOneItem</td><td>
Save the specific value among PT items in the RAM of the drive.

Sending : 8 bytes
<table>
<tr><td>2 bytes</td><td>2bytes</td><td>4 bytes</td></tr>
<tr><td>PT No.(0~255) to be saved</td><td>Offset value(0~40) of the specific items to be saved</td><td>The value to be saved</td></tr>
</table>
Refer to 「1-2-6. Position Table Item」 for offset value.

Response : 2 bytes
<table>
<tr><td>1 byte</td><td>1 byte</td></tr>
<tr><td>Communication Status</td><td>Command performance (values except 0 : complete, 0: error))</td></tr>
</table>
</td></tr>
<tr><td>0x78
(120)</td><td>FAS_
MovePush</td><td>
Start push motion which maintains a fixed force.

Sending : 28 bytes
<table>
<tr><td>4 bytes</td><td>4 bytes</td><td>4 bytes</td><td>2 bytes</td><td>2 bytes</td></tr>
<tr><td>Position movement start speed value</td><td>Position movement speed value</td><td>Position movement absolute position value</td><td>Position movement acceleration time</td><td>Position movement deceleration time</td></tr>
</table>
<table>
<tr><td>2 bytes</td><td>4 bytes</td><td>4 bytes</td><td>2 bytes</td></tr>
<tr><td>Push motion torque ratio</td><td>Push motion movement speed</td><td>Push motion absolute position value</td><td>Push mode</td></tr>
</table>
Position movement start speed value : 1~35,000[pps]
Position movement speed value : 1~500,000[pps]
Position movement absolute position value : -134,217,728~134,217,727
Position movement acc/dec time : 1~9,999[ms]
Push motion torque ration : 20~90[%]
</td></tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td>Push motion movement speed : 1~33,333[pps] (Max. 200[rpm]) (Resolution : 10,000)
Push motion absolute position value : -134,217,728~134,217,727
push mode : 0(stop mode), 1~10,000(non-stop mode)

Refer to 「User Manual Text 8-6. Push Motion Function」 for more details.

Response : 1 byte
<table><tr><td>1 byte</td></tr><tr><td>Communication status</td></tr></table>
◆ Must run Stop(E-Stop) command before any motion commands.</td>
</tr>
<tr>
<td>0x79
(121)</td>
<td>FAS_
GetPushStatus</td>
<td>Confirm the current push motion status.

Sending : 0 byte
Response : 1 byte
<table><tr><td>1 byte</td><td>1 byte</td></tr><tr><td>Communication status</td><td>Push motion status
0: Normal position move wait status
1: In push motion, not yet contact in work.
2: In contact with work, power is maintained.
3: Push motion is completed but not in contact with work . In this case, the Stop (E-Stop) command must be executed before the next motion command.</td></tr></table></td>
</tr>
</table>

- Frame Type '0x65 ~ 0x69', '0x90 ~ 0x92' are allotted for internal use.

1 - 2 - 2 . Parameter Lists

No.	Name	Unit	Min	Max	Default
0	Pulse Per Revolution		0	8	8
1	Axis Max Speed	[pps]	1	500,000	500,000
2	Axis Start Speed	[pps]	1	35,000	1
3	Axis Acc Time	[msec]	1	9,999	100
4	Axis Dec Time	[msec]	1	9,999	100
5	Speed Override	[%]	1	500	100
6	Jog Speed	[pps]	1	500,000	5,000
7	Jog Start Speed	[pps]	1	500,000	1
8	Jog Acc Dec Time	[msec]	1	9,999	100
9	S/W Limit Plus Value	[pulse]	-134,217,728	134,217,727	134,217,727
10	S/W Limit Minus Value	[pulse]	-134,217,728	134,217,727	-134,217,728
11	S/W Limit Stop Method		0	2	2
12	H/W Limit Stop Method		0	1	0
13	Limit Sensor Logic		0	1	0
14	Org Speed	[pps]	1	500,000	5,000
15	Org Search Speed	[pps]	1	50,000	1,000
16	Org Acc Dec Time	[msec]	1	9,999	50
17	Org Method		0	7	0
18	Org Dir		0	1	1
19	Org OffSet	[pulse]	-134,217,728	134,217,727	0
20	Org Position Set	[pulse]	-134,217,728	134,217,727	0
21	Org Sensor Logic		0	1	0
22	Position Loop Gain		0	127	4
23	Inpos Value		0	63	0
24	Pos Tracking Limit	[pulse]	1	134,217,727	5,000
25	Motion Dir		0	1	0
26	Limit Sensor Dir		0	1	0
27	Org Torque Ratio	[%]	20	90	50
28	Pos. Error Overflow Limit	[pulse]	1	134,217,727	5,000
29*1	Brake Delay Time	[msec]	10	5,000	200
30	Run Current	*10[%]	5	15	10
31	Boost Current	*50[%]	0	7	0
32	Stop Current	*10[%]	2	10	5
33	JOG EXT FUNC USE		0	1	0
34	Jog Speed1	[pps]	1	500,000	5,000
35	Jog Speed2	[pps]	1	500,000	5,000
36	Jog Speed3	[pps]	1	500,000	5,000
37	Jog Speed4	[pps]	1	500,000	5,000
38	Jog Speed5	[pps]	1	500,000	5,000

39	Jog Speed6	[pps]	1	500,000	5,000
40	Jog Speed7	[pps]	1	500,000	5,000
41	Use Motion Queue		0	1	0
42	Disconnection Option		0	4	0
43	Communication Timeout	msec	100	60,000	100
44	Motion Profile		0	1	0

*1 This parameter is not used for drives for 86[mm] motors.

- Parameter No. 44 is available for Firmware [ver.6.1.20.18] or higher version.

1 - 2 - 3 . Bit setup of Output pin

Detailed description of frame type '0x20'.

This command is applicable only to 9 signals from 「User Output 0」 to 「User Output 8」 out of 24 signal types of control output. The remaining 15 output signals cannot be manipulated by the user. When the situation occurs during drive operation, the corresponding signals are output.

The following table shows the bit mask values for each signal.

Signal Name	Corresponding Bit Position	Signal Name	Corresponding Bit Position	Signal Name	Corresponding Bit Position
Compare Out	0x00000001	Origin Search OK	0x00000100	User OUT 1	0x00010000
Inposition	0x00000002	ServoReady	0x00000200	User OUT 2	0x00020000
Alarm	0x00000004	reserved	0x00000400	User OUT 3	0x00040000
Moving	0x00000008	Brake	0x00000800	User OUT 4	0x00080000
Acc/Dec	0x00000010	PT Output0	0x00001000	User OUT 5	0x00100000
ACK	0x00000020	PT Output1	0x00002000	User OUT 6	0x00200000
END	0x00000040	PT Output2	0x00004000	User OUT 7	0x00400000
AlarmBlink	0x00000080	User OUT 0	0x00008000	User OUT 8	0x00800000

【Example 1】 Sending data to turn ON the User Output 5 port.

4 bytes (I/O set mask value)	4 bytes (I/O clear mask value)
0x00100000	0x00000000

【Example 2】 Sending data to turn OFF the User Output 5 port

4 bytes (I/O set mask value)	4 bytes (I/O clear mask value)
0x00000000	0x00100000

1 - 2 - 4 . Bit setup of Input pin

Detailed description of the Frame type at 0x21.

This command is applied to 32 signals at the control input. It can be used for testing as if the signal was input without the actual input signal.

The following table shows the bit mask values for each signal.

Signal Name	Corresponding Bit Position	Signal Name	Corresponding Bit Position	Signal Name	Corresponding Bit Position	Signal Name	Corresponding Bit Position
Limit+	0x00000001	PT A4	0x00000100	Alarm Reset	0x00010000	JPT input2	0x01000000
Limit-	0x00000002	PT A5/ User IN 6/ Jog0	0x00000200	ServoON	0x00020000	JPT Start	0x02000000
Origin	0x00000004	PT A6/ User IN 7/ Jog1	0x00000400	Pause	0x00040000	User IN 0	0x04000000
Clear Position	0x00000008	PT A7/ User IN 8/ Jog2	0x00000800	Org Search	0x00080000	User IN 1	0x08000000
PT A0	0x00000010	PT Start	0x00001000	Teaching	0x00100000	User IN 2	0x10000000
PT A1	0x00000020	Stop	0x00002000	E-stop	0x00200000	User IN 3	0x20000000
PT A2	0x00000040	Jog+	0x00004000	JPT input0	0x00400000	User IN 4	0x40000000
PT A3	0x00000080	Jog-	0x00008000	JPT input1	0x00800000	User IN 5	0x80000000

【Example 1】 Sending data to turn ON the Pause port

4 bytes (I/O set mask value)	4 bytes (I/O clear mask value)
0x00040000	0x00000000

【Example 2】 Sending data to turn OFF the Pause port

4 bytes (I/O set mask value)	4 bytes (I/O clear mask value)
0x00000000	0x00040000

Caution	**You cannot use the functions of 'PT A5 ~ PT A7' and 'User IN6 ~ IN8' at the same time.**

1 - 2 - 5 . Bit setup of Status Flag

Refer to 'MOTION_EziSERVO2_DEFINE.h' of include files.

Name of Flag Define	Contents	Corresponding Bit Position
FFLAG_ERRORALL	One or more error occurs.	0X00000001
FFLAG_HWPOSILMT	'+' direction limit sensor turns ON.	0X00000002
FFLAG_HWNEGALMT	'-' direction limit sensor turns ON.	0X00000004
FFLAG_SWPOGILMT	'+' direction program limit is exceeded.	0X00000008
FFLAG_SWNEGALMT	'-' direction program limit is exceeded.	0X00000010
Reserved1		0X00000020
Reserved2		0X00000040
FFLAG_ERRPOSOVERFLOW	The position error is bigger than 'Pos Error Overflow Limit' after completing position command.	0X00000080
FFLAG_ERROVERCURRENT	The motor driving device is under over-current.	0X00000100
FFLAG_ERROVERSPEED	The motor speed exceeded 3000[rpm].	0X00000200
FFLAG_ERRPOSTRACKING	If the position error is bigger than 'Pos Tracking Limit' during position command	0X00000400
FFLAG_ERROVERLOAD	Alarm occurs when a load exceeding the maximum torque of the motor is applied for a distance of more than 5 seconds or more than 10 revolutions.	0X00000800
FFLAG_ERROVERHEAT	The internal temperature of the drive exceeds 85°C.	0X00001000
FFLAG_ERRBACKEMF	A counter electromotive force of the motor exceeds 70V.	0X00002000
FFLAG_ERRMOTORPOWER	There is a connection error with the motor.	0X00004000
FFLAG_ERRINPOSITION	Alarm is generated when Inposition error occurs.	0X00008000
FFLAG_EMGSTOP	The motor is under emergency stop.	0X00010000
FFLAG_SLOWSTOP	The motor is under general stop.	0X00020000
FFLAG_ORIGINRETURNING	The motor is returning to the origin.	0X00040000
FFLAG_INPOSITION	Inposition is finished.	0X00080000
FFLAG_SERVOON	The motor is Servo ON.	0X00100000
FFLAG_ALARMRESET	Alarm Reset command is executed.	0X00200000
FFLAG_PTSTOPED	Position Table operation has been finished.	0X00400000
FFLAG_ORIGINSENSOR	The origin sensor is ON.	0X00800000
FFLAG_ZPULSE	The motor is in the z-pulse position of encoder.	0X01000000
FFLAG_ORIGINRETOK	Origin return operation has been finished.	0X02000000
FFLAG_MOTIONDIR	To display the motor operating direction (+: Off, -: On)	0X04000000
FFLAG_MOTIONING	The motor is running.	0X08000000
FFLAG_MOTIONPAUSE	The motor in running is stopped by Pause command.	0X10000000
FFLAG_MOTIONACCEL	The motor is operating to the acceleration	0X20000000

	section.	
FFLAG_MOTIONDECEL	The motor is operating to the deceleration section.	0X40000000
FFLAG_MOTIONCONST	The motor is operating at constant speed.	0X80000000

1 - 2 - 6 . Position Table Item

Refer to 'motion_define.h' of include files.

Name	Name of Structure Parameter	Number of Bytes	Offset value	Unit	Low Limit	Upper Limit
Position	lPosition	4 (signed)	0	[pulse]	-134,217,728	+134,217,727
Low Speed	dwStartSpd	4 (unsigned)	4	[pps]	0	500,000
High Speed	dwMoveSpd	4 (unsigned)	8	[pps]	0	500,000
Accel. Time	wAccelRate	2 (unsigned)	12	[msec]	1	9,999
Decel. Time	wDecelRate	2 (unsigned)	14	[msec]	1	9,999
Command	wCommand	2 (unsigned)	16		0	10
Wait time	wWaitTime	2 (unsigned)	18	[msec]	0	600,000
Continuous Action	wContinuous	2 (unsigned)	20		0	1
Jump Table No.	wBranch	2 (unsigned)	22		0 10,000	255 10,255
Jump PT 0	wCond_branch0	2 (unsigned)	24		0 10,000	255 10,255
Jump PT 1	wCond_branch1	2 (unsigned)	26		0 10,000	255 10,255
Jump PT 2	wCond_branch2	2 (unsigned)	28		0 10,000	255 10,255
Loop Count	wLoopCount	2 (unsigned)	30		0	100
Loop Jump Table No.	wBranchAfterLoop	2 (unsigned)	32		0 10,000	255 10,255
PT set	wPTSet	2 (unsigned)	34		0	15
Loop Counter Clear	wLoopCountCLR	2 (unsigned)	36		0	255
Check Inposition	bCheckInpos	2 (unsigned)	38		0	1
Compare Position	lTriggerPos	4 (signed)	40	[pulse]	-134,217,728	+134,217,727
Compare Width	wTriggerOnTime	2 (unsigned)	44	[msec]	1	9,999
Push Ratio	wPushRatio	2 (unsigned)	46	[%]	20	90
Push Speed	dwPushSpeed	4 (unsigned)	48	[pps]	0	33,333
Push Position	lPushPosition	4 (signed)	52	[pulse]	-134,217,728	+134,217,727
Push Mode	wPushMode	2 (unsigned)	56		0	10,000

Blank		6 (unsigned)	58	0x00

For the setting method of each item, refer to other manual 「User Manual_Position Table」

1 - 3 . Program Method

There are two ways of programming to use Ezi-SERVOⅡ Plus-E ALL.

The first is to use the Visual C ++ language in the window system of a PC.
In this case, use the library supplied with the product (see 「2. Library for PC Program」).

The second method is to send a command character directly without using a library function. Like protocol test program, user should write low level protocol program by himself / herself. It is mainly applied when PLC is used as host controller.

2. Library for PC Program

2 - 1 . Library Configuration

(1) For C++

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 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"

The library files are as follows:
1) For 32bit
"₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include₩₩EziMotionPlusE.lib"
"₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include₩₩EziMotionPlusE.dll"

2) For 64bit
"₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include_x64₩₩EziMotionPlusE.lib"
"₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include_x64₩₩EziMotionPlusE.dll"

Sample program source using this library is included in
"₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩Examples₩₩C++₩₩"

(2) For C#

To use this library, C# header file and library file are required. These files are included in "₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩" The following contents should be included in a source file for development.

#include "₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include₩₩MOTION_DEFINE_PlusE.cs

The library files are as follows:
1) For 32bit
"₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include₩₩LIB_EziMOTIONPlusE.cs"

2) For 64bit
"₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩include_x64₩₩LIB_EziMOTIONPlusE.cs"

Sample program source using this library is included in

"₩₩FASTECH₩₩Ezi-MOTION Plus-E V6₩₩Examples₩₩C#₩₩"

(3) The following table describes the values returned when using each library function. These return values can only be seen in library (DLL) functions and are not supported in programs using the protocol.

Type	Name	Return value	Description
Normal	FMM_OK	0(0x00)	The function has normally performed the command.
Input Error	FMM_NOT_OPEN	1(0x01)	Wrong port number is input.
	FMM_INVALID_PORT_NUM	2(0x02)	Unconnected port number
	FMM_INVALID_SLAVE_NUM	3(0x03)	Wrong slave number is input.
Operation Error	FMM_POSTABLE_ERROR	9(0x09)	An error occurred while reading / writing the position table.
Connection Error	FMC_DISCONNECTED	5(0x05)	The corresponding drive is disconnected.
	FMC_TIMEOUT_ERROR	6(0x06)	There is no response for the specified time. (100 msec)
	FMC_RECVPACKET_ERROR	8(0x08)	A protocol level error has occurred in a packet received from the drive.

(4) The following table shows the return values that are common to all libraries. You can check the results (communication status and operation status) checked by the drive. It is supported for both using libraries (DLLs) and programming using protocols.

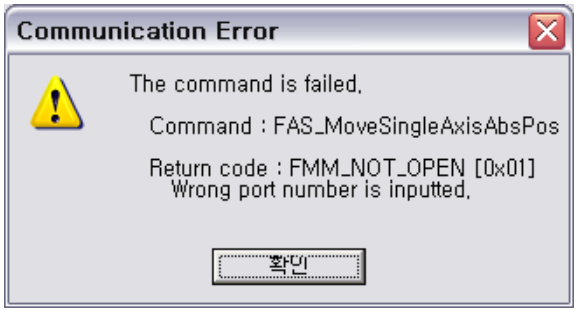
Type	Name	Return value	Description
Normal	FMP_OK	0(0x00)	Communication has been normally performed.
Input Error	FMP_FRAMETYPEERROR	128(0x80)	The drive cannot recognize the command.
	FMP_DATAERROR	129(0x81)	Input data is out of the range.
Operation Error	FMP_RUNFAIL	133(0x85)	Incorrect command, such as the motor is already running or not ready to run.
	FMP_RESETFAIL	134(0x86)	Alarm Reset command cannot be executed in Servo ON.
	FMP_SERVOONFAIL1	135(0x87)	Alarm has occurred.
	FMP_SERVOONFAIL2	136(0x88)	The motor is under Emergency Stop.
	FMP_SERVOONFAIL3	137(0x89)	'Servo ON' is already set for the input signal.
Connectio	FMP_PACKETERROR	130(0x82)	A protocol level error has occurred in a packet received by the drive.

n Error			

2 - 2 . Communication status Window

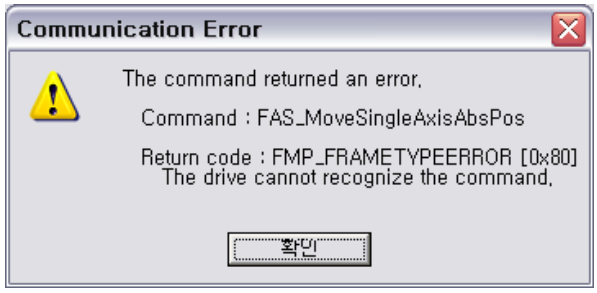
Communication status can be classified into three categories.

(1) Communication Error



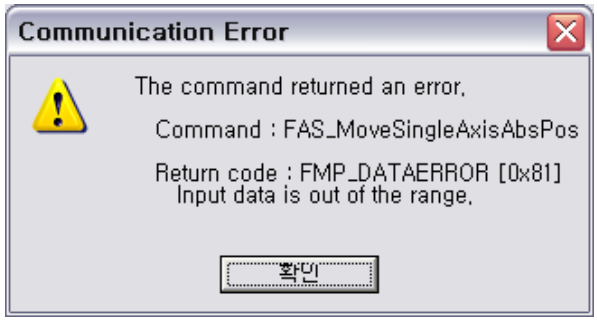
FMM_NOT_OPEN,

COM port is not connected. (This error cannot occur in the GUI.)



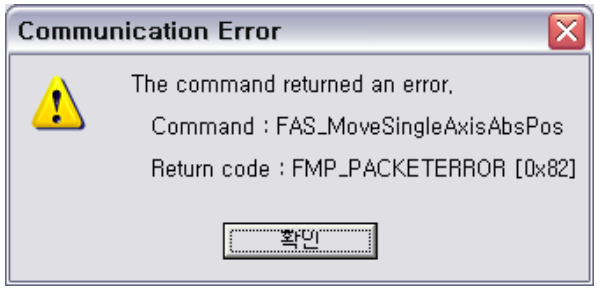
FMP_FRAMETYPEERROR = 0x80,

Drive did not recognize the command or the command is not supported.



FMP_DATAERROR,

The data input is out of the range supported by Drive.



FMP_PACKETERROR,

The received frame is data that does not meet the standard. (The length of Packet sent to Drive does not match.)

(2) Wrong Command

FMP_RUNFAIL

Drive Command Failed: Attempted to execute a new drive under the following conditions.

-The motor is currently running

-During stop command

-Servo OFF status

-Try Z-pulse Origin without external encoder

-Other invalid driving commands

FMP_RESETFAIL,

Attempt to execute Reset in the following conditions.

-Servo ON status

-Already Reset by external input signal.

FMP_SERVOONFAIL1,

An attempt was made to execute a Servo ON command while an alarm occurred.

FMP_SERVOONFAIL2,

An attempt was made to execute a Servo Off command during an emergency stop.

FMP_SERVOONFAIL3,

The Servo ON signal is assigned to an external input and cannot be executed.

(3) Command Execution Error

FMM_POSTABLE_ERROR

,

Failed to execute function related to position table.

2 - 3 . Drive Connection Function

Function Name	Description
FAS_Connect	Attempt to connect with the drive using the UDP Protocol. : Returns TRUE if the connection was successful and FALSE if the connection failed.
FAS_ConnectTCP	Attempt to connect with the drive using the TCP Protocol. : Returns TRUE if the connection was successful and FALSE if the connection failed.
FAS_Reconnect	Reconnect with existing IP, Protocol, and iBdID.
FAS_AutoReconnect	In case of using TCP, if the response is not within 100 [ms] or if TCP is disconnected unintentionally, it automatically connects to another port and executes the unreceived function again.
FAS_Close	Attempt to terminate communication with the drive.
FAS_GetSlaveInfo	Reads drive type and program version. : Returns drive type and version information.
FAS_GetMotorInfo	Reads the type and manufacturer of the motor connected to the drive.
FAS_IsSlaveExist	Check the existence of the drive. : Returns TRUE if present, FALSE if connection failed.
FAS_IsBdIDExist	Check whether BdID is used for the IP address. : Returns TRUE if present, FALSE if connection failed.
FAS_IsIPAddressExist	Check whether the IP Address is assigned to the corresponding BdID. : Returns TRUE if present, FALSE if connection failed.
FAS_EnableLog	Controls output of log related to communication error. : Returns TRUE if present, FALSE if connection failed.
FAS_SetLogPath	Set the path to save the output log. : Returns TRUE if present, FALSE if connection failed.
FAS_SetLogLevel	Output log according to the set level. : By default, only the logs related to internal communication errors are displayed. (LOG_LEVEL_COMM)
FAS_PrintCustomLog	Output arbitrary log.

● 1. **The following functions are supported by F / W Ver V06.01.020.04 Library Ver 2.0.0.10 or later.**

1) FAS_ConnectTCP

2) FAS_Reconnect

3) FAS_SetLogLevel

4) FAS_PrintCustomLog

2. **The following functions are supported by F / W Ver V06.01.020.05 Library Ver 2.3.0.15 or later.**

1) FAS_SetAutoReconnect

FAS_Connect

FAS_Connect is a function that connects Ezi-SERVOⅡ Plus-E ALL with UDP Protocol.

Syntax

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

Parameters

sb1~4

Enter the IP address of the drive you want to connect to.

ex) 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.

You can not use the same ID as an IP address.

Return Value

Returns TRUE if the connection was successful and FALSE if the connection failed.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInit()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0 // The 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("Connect fail!"));
                return;
```

```
        }

        if (FAS_IsSlaveExist(iBdID) == FALSE)
        {
        {
                // The board number does not exist.
                // Check the board number of Ezi-SERVOⅡ.
                return;
        }

        nRtn = FAS_GetSlaveInfo(iBdID, &nType, lpBuff, nBuffSize);
        if (nRtn != FMM_OK)
        {
                // The command did not run successfully.
                // Refer to ReturnCodes_Define.h
        }

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

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

See Also

FAS_Close

FAS_ConnectTCP

FAS_Connect is a function that connects Ezi-SERVOⅡ Plus-E ALL with TCP Protocol.

Syntax

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

Parameters

sb1~4

Enter the IP address of the drive you want to access.

ex) For 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.

You can not use the same ID as an IP address.

Return Value

Returns TRUE if the connection was successful and FALSE if the connection failed.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInit()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0 // 192.168.0.2 의 Board 고유 번호
        char lpBuff[256];
        int nBuffSize = 256;
        BYTE nType;
        int nRtn;

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

```
        }

        if (FAS_IsSlaveExist(iBdID) == FALSE)
        {
                // The board number does not exist.
                // Check the board number of Ezi-SERVOⅡ.
                return;
        }

        nRtn = FAS_GetSlaveInfo(iBdID, &nType, lpBuff, nBuffSize);
        if (nRtn != FMM_OK)
        {
                // The command did not run successfully.
                // Refer to ReturnCodes_Define.h
        }

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

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

See Also

FAS_Close

FAS_Reconnect

Reconnect to the protocol you were using without FAS_Connect.

Syntax

```
void FAS_Reconnect(int iBdID);
```

Parameters

iBdID

Drive ID number to reconnect.

Remarks

After connecting with FAS_Connect () function, when the connection is terminated or terminated, this command connects communication without using FAS_Connect () again.

Example

Refer to FAS_Connect library.

See Also

FAS_Connect

FAS_SetAutoReconnect

Automatically connect the TCP communication.

Syntax

void FAS_SetAutoReconnect(BOOL bSET);

Parameters

bSET

Set whether to use the Auto Reconnect function.

Remarks

After setting FAS_SetAutoReconnect to Set, if the response is not within 100 [ms] or the TCP is disconnected unintentionally, connect using a different port and execute the unresponsive function again.

(Only executed when connected using FAS_ConnectTCP.)

- When connecting to the GUI while using the above functions, be sure to connect the GUI by UDP.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcInit()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0 // Drive number of 102.168.0.2
        char lpBuff[256];
        int nBuffSize = 256;
        BYTE nType;
        int nRtn;

        // Try to connect
        if (FAS_ConnectTCP(sb1, sb2, sb3, sb4, iBdID) == FALSE)
        {
                // Connection failed.
                MessageBox(_T("connect fail!"));
                return;
        }
        // Enable Auto Reconnect
        FAS_SetAutoReconnect(SET);
}
```

See Also

FAS_Close

Disconnect communication of the ID.

Syntax

```
void FAS_Close(int iBdID);
```

Parameters

iBdID

Drive ID number to disconnect.

Remarks

Example

Refer to FAS_Connect library

See Also

FAS_Connect

FAS_GetSlaveInfo

Get version information string of the relevant drive.

Syntax

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

Parameters

iBdID

ID number of the drive. IBdID set in FAS_Connect function

pType

Type number of the drive.

lpBuff

Buffer Pointer to receive Version information string.

nBuffSize

Memory allocation size value of lpBuff.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

Refer to FAS_Connect library

See Also

FAS_GetMotorInfo

Gets the information string of the motor connected to the drive.

Syntax

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

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

pType

Type number of the drive.

lpBuff

Buffer Pointer to receive Version information string.

nBuffSize

Memory allocation size value of lpBuff.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

Refer to FAS_Connect library

See Also

FAS_IsSlaveExist

Check if the drive is currently connected.

Syntax

```
BOOL FAS_IsSlaveExist(int iBdID);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

Return Value

TRUE : Connected.

FALSE : Disconnected.

Remarks

This function is provided only in the library(DLL) and is not supported for protocol programming.

Example

Refer to FAS_Connect library

See Also

FAS_Connect

FAS_IsBdIDExist

Check whether the drive is currently connected.

Syntax

```
BOOL FAS_IsBdIDExist(int iBdID, BYTE* sb1, BYTE* sb2, BYTE* sb3, BYTE* sb4 );
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

Return Value

TRUE : The corresponding BdID is used.

FALSE : The corresponding BdID is not used.

Remarks

This function is provided only in the library(DLL) and is not supported for protocol programming.

Example

Refer to FAS_Connect library

See Also

FAS_Connect

FAS_IsIPAddressExist

Check if the drive is currently connected.

Syntax

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

Parameters

sb1, sb2, sb3, sb4

IP Address. (ex,192.168.0.10 → sb1:192, sb2:168, sb3:0, sb4:10)

iBdID

ID number of the drive. IBdID set by the FAS-Connect function.

Return Value

TRUE : The corresponding IP Address is used.

FALSE : The corresponding IP Address is not used.

Remarks

This function is provided only in the library(DLL) and is not supported for protocol programming.

Example

Refer to FAS_Connect library

See Also

FAS_Connect

FAS_EnableLog

Determines whether to output the communication error related log.

Syntax

```
void FAS_EnableLog(BOOL bEnable);
```

Parameters

bEnable

Set whether to output the log.

Remarks

Controls log output that occurs while using Ezi-MOTION Plus-E function in the current process.
This setting does not affect the log output of other processes or programs.
Log is generated from FAS_Connect.
When FAS_Close is used to disconnect the currently connected drive, log output is terminated. The default setting for Log output is TRUE.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcDisableLog()
{
        FAS_EnableLog(FALSE);

        // After this, the log of the functions will not be output.

        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0 // Drive number of 102.168.0.2

        // 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

Set the path to save the output log.

Syntax

BOOL FAS_SetLogPath(LPCTSTR lpPath);

Parameters

lpPath

Folder path Character string of Log output file.

Return Value

If the path entered does not exist or cannot be accessed, FALSE is returned.

Remarks

This function must be called before using the FAS_Connect function.

If the lpPath value is NULL or a zero-length string is entered, the log path is set to the folder where the program currently using Ezi-MOTION Plus-E Library exists.

The default value for the log path is NULL, which is the folder where the current program exists.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcEnableLog()
{
	BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
	int iBdID = 0 // Drive number of 102.168.0.2

	// Output Log.
	FAS_EnableLog(TRUE);   // No need to use

	if (!FAS_SetLogPath(_T("C:₩₩Logs₩₩"))) // C:₩Logs folder should exist.
	{
		// Log path does not exist.
		Return;
	}

	// Logs of all functions are saved in the 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

FAS_SetLogLevel

Set the range of the log to be output.

Syntax

BOOL FAS_SetLogLevel(enum LOG_LEVEL level);

Parameters

level

Log output range setting

Return Value

If a value other than the level setting value is entered, FALSE is returned.

Remarks

LOG_LEVEL_COMM : Only logs related to communication errors are displayed.
LOG_LEVEL_PARAM : The parameter setting function log is additionally output to the above log output.
LOG_LEVEL_MOTION : Motion command function log is additionally output to the above log output.
LOG_LEVEL_ALL : All logs that can be output are displayed.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcEnableLog()
{
	BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
	int iBdID = 0 // Drive number of 102.168.0.2

	// Output Log.
	FAS_EnableLog(TRUE);   // No need to use.

	FAS_SetLogLevel(LOG_LEVEL_ALL); // Set the log output range.

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

		return;
	}

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

See Also

FAS_EnableLog

FAS_PrintCustomLog

Output arbitrary log.

Syntax

```
BOOL FAS_PrintCustomLog(
        int iBdID,
        enum LOG_LEVEL level,
        LPCTSTR lpszMsg
        );
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

level

Log output range setting

lpszMsg

String of log to be output

Return Value

If a value other than the level setting value is entered, FALSE is returned.

Remarks

Level is the same as the set value (range) of FAS_SetLogLevel ().

Used to print the log at a specific location (function) in the program

Or, it is used to log output differently from the setting value of FAS_SetLogLevel () in the program.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcCustomLog()
{
        int iBdID = 0 // Drive number of 102.168.0.2
        int level = LOG_LEVEL_PARAM;

       // Communication error and parameter setting function log output setting
       FAS_PrintCustomLog (iBdID, level, lpszMsg );

}
```

See Also

FAS_ SetLogLevel

2 - 4 . Parameter Control Function

Function Name	Description
FAS_SaveAllParameters	Save current status parameters to ROM : Saves parameters such as operation speed, acc / dec time and home return to be saved even after the drive is turned off.
FAS_SetParameter	Save the designated parameter value in RAM area : Save the specific parameter value.
FAS_GetParameter	Read the specified parameter value from RAM area : Read specific parameter value.
FAS_GetROMParameter	Read the specified parameter value from ROM area : Read specific parameter value in ROM area.

FAS_SaveAllParameters

Saves all modified parameter values and I / O signal assignment values in the ROM area.

Syntax

```
Int FAS_SaveAllParameters(
    int iBdID
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

In addition to the current parameter values, the values set in the 'FAS_SetIOAssignMap' library are also stored in the ROM memory.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

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

        long lParamVal;
        int nRtn;

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

                return;
        }

        // Check Axis Start Speed Parameter.
        nRtn = FAS_GetParameter(iBdID, SERVO_AXISSTARTSPEED, &lParamVal);
        if (nRtn != FMM_OK)
        {
                // The command did not run successfully.
                // Refer to ReturnCodes_Define.h
                _ASSERT(FALSE);
        }
        else
        {
                // This is the parameter value saved in Ezi-SERVOⅡ.
```

```
            printf("Parameter [before] : Start Speed = %d ₩n", lParamVal);
        }


        // Change Start Speed Parameter value to 200 and re-read the value.
        nRtn = FAS_SetParameter(iBdID, SERVO_AXISSTARTSPEED, 200);
        _ASSERT(nRtn == FMM_OK);   // If the command did not complete successfully, stop.

        nRtn = FAS_GetParameter(iBdID, SERVO_AXISSTARTSPEED, &lParamVal);
        _ASSERT(nRtn == FMM_OK);
        printf("Parameter [after] : Start Speed = %d ₩n", lParamVal);


        // Check the value saved in the ROM.
        nRtn = FAS_GetROMParameter(iBdID, SERVO_AXISSTARTSPEED, &lParamVal);
        _ASSERT(nRtn == FMM_OK);   // If the command did not complete successfully, stop.
        printf("Parameter [ROM] : Start Speed = %d ₩n", lParamVal);


        // After chainging parameter value, save it to ROM.
        nRtn = FAS_SetParameter(iBdID, SERVO_AXISSTARTSPEED, 100);
        _ASSERT(nRtn == FMM_OK);   // If the command did not complete successfully, stop.

        nRtn = FAS_SaveAllParameters(iBdID);
        _ASSERT(nRtn == FMM_OK);

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

See Also

FAS_GetRomParameter

FAS_SetParameter

Modify the parameter value (store it in the RAM area).

Syntax

```
int FAS_SetParameter(
    int iBdID,
    BYTE iParamNo,
    long lParamValue
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function. .

iParamNo

Parameter No. to be modified.

lParamValue

Parameter value to be modified.

Return Value

FMM_OK : The command ran successfully.
FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.
FMM_INVALID_PARAMETER_NUM: The parameter of the specified iParamNo does not exist

Remarks

Applies only to one specified parameter.
The parameters on the drive are stored in two memory areas. In other words, the parameters are permanently stored in the ROM to be maintained when the power is turned off, and the ROM parameters are copied to the DSP's RAM when the power is turned on.
When the user changes the parameters, the parameters in the ROM are not changed, but the parameters in RAM are changed.
This function sets the parameter of the specified number in RAM to the desired value.

Example

Reger to FAS_SaveAllParameter library

See Also

FAS_GetParameter

FAS_GetParamater

Load specific parameter of Drive.

Syntax

```
int FAS_GetParameter(
    int iBdID,
    BYTE iParamNo,
    long* lParamValue
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function. .

iParamNo

Parameter No. to be loaded.

lParamValue

Parameter value.

Return Value

FMM_OK : The command ran successfully.
FMM_NOT_OPEN : The drive it not connected yet.
FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.
FMM_INVALID_PARAMETER_NUM: The parameter of the specified iParamNo does not exist.

Remarks

Applies only to one specified parameter.
The parameters on the drive are stored in two memory areas. In other words, the parameters are permanently stored in the ROM to be maintained when the power is turned off, and the ROM parameters are copied to the DSP's RAM when the power is turned on.
When the user changes the parameters, the parameters in the ROM are not changed, but the parameters in RAM are changed.
This function loads the parameter of the specified number in RAM.

Example

Refer to FAS_SaveAllParameter

See Also

FAS_SetParameter

FAS_GetROMParameter

Load the parameter saved in the ROM area.

Syntax

```
int FAS_GetROMParameter(
    int iBdID,
    BYTE iParamNo,
    long* lRomParam
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

iParamNo

Parameter No. to be loaded.

lRomParam

Parameter value saved in ROM.

Return Value

FMM_OK : The command ran successfully.
FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.
FMM_INVALID_PARAMETER_NUM: The parameter of the specified iParamNo does not exist

Remarks

Load parameter values saved in ROM.
Executing this function does not change the value in RAM. To do this, run FAS_SetParameter.

Example

Refer to FAS_SaveAllParameter library.

See Also

FAS_SaveAllParameters

2 - 5 . Servo Control Function

Function name	Describtion
FAS_ServoEnable	Turns the step status of the specified drive ON / OFF.
FAS_ServoAlarmReset	Release the alarm of the drive where the alarm occurred : Remove the cause of the alarm and execute it.
FAS_GetAlarmType	Check the current alarm occurrence and the type of alarm.

FAS_ServoEnable

Turn the drive Servo ON / OFF.

Syntax

```
int FAS_ServoEnable(
    int iBdID,
    BOOL bOnOff
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

bOnOff

Enable or Disable.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

After enabling, it takes some time for Servo (Step) ON flag of Axis Status to be 'ON'.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcAxisStatus()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0;          //   Drive number
        EZISERVO_AXISSTATUS AxisStatus;
        int nRtn;

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

        nRtn = FAS_GetAxisStatus(iBdID, &(AxisStatus.dwValue));
```

```
        _ASSERT(nRtn == FMM_OK);

        // Servo(Step) On when SERVO_ON flag is OFF.
        if (AxisStatus.FFLAG_SERVOON == 0)
        {
                nRtn = FAS_ServoEnable(iBdID, TRUE);
                _ASSERT(nRtn == FMM_OK);
        }

        // AlarmReset when Alarm is active.
        if (AxisStatus.FFLAG_ERRORALL || AxisStatus.FFLAG_ERROVERCURRENT ||
AxisStatus.FFLAG_ERROVERLOAD)
        {
                nRtn = FAS_ServoAlarmReset(iBdID);
                _ASSERT(nRtn == FMM_OK);
        }

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

See Also

FAS_ServoAlarmReset

FAS_ServoAlarmReset

Send an AlarmReset command.

Syntax

```
int FAS_ServoAlarmReset(
    int iBdID
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Before sending this command, remove the cause of the alarm first.

Refer to 'User Manual_Text' for the cause of the alarm.

Example

Refer to FAS_ServoEnable library

See Also

FAS_ServoEnable

2 - 6 . Control I/O Function

Function Name	Description
FAS_SetIOInput	Set input signal level of control input : Sets the input signal to [ON] or [OFF].
FAS_GetIOInput	Read current input signal status of control input : Returns in bit unit for each input signal.
FAS_SetIOOutput	Sets output signal level of control output : Sets the output signal to [ON] or [OFF].
FAS_GetIOOutput	Read current output signal status of control output : Returns in bit unit for each output signal.
FAS_GetIOAssignMap	The CN1 port pin setting status is read. : Returns the setting status of the signal that can be set in the input and output ports in bit units.
FAS_SetIOAssignMap	Assign control I / O signal to pin of CN1 port and set signal level : Set the signal that can be set with variable input and output ports.
FAS_IOAssignMapReadROM	The setting status and signal level status of control input and output port are read from ROM area to RAM area.

FAS_SetIOInput

Set the I/O input. For more information, refer to '1-2. Structure of Frame Type'.

Syntax

```
int FAS_SetIOInput(
    int iBdID,
    DWORD dwIOSetMask,
    DWORD dwIOCLRMask
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

dwIOSetMask

Bitmask value of Input when Set(ON).

dwIOCLRMask

Bitmask value of Input when Clear(OFF).

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Be careful not to duplicate the bit values of dwIOSetMask and dwIOCLRMask.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcIO()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0;          //   Drive number
        DWORD dwInput, dwOutput;
        int nRtn;

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

                return;
```

```
		}

		// Check I/O Input.
		nRtn = FAS_GetIOInput(iBdID, &dwInput);
		_ASSERT(nRtn == FMM_OK);
		if (dwInput & SERVO_IN_BITMASK_LIMITP)
		{
			// Limit+ input is ON.
		}

		if (dwInput & SERVO_IN_BITMASK_USERIN0)
		{
			// User Input 0 is ON.
		}

		// Clear Position and User Input 1 is ON and Jog+ Input is OFF.
		nRtn = FAS_SetIOInput(iBdID, SERVO_IN_BITMASK_CLEARPOSITION |
SERVO_IN_BITMASK_USERIN1, SERVO_IN_BITMASK_PJOG);
		_ASSERT(nRtn == FMM_OK);

		// Check the I/O Output.
		nRtn = FAS_GetIOOutput(iBdID, &dwOutput);
		_ASSERT(nRtn == FMM_OK);
		if (dwOutput & SERVO_OUT_BITMASK_USEROUT0)
		{
			// User Output 0 signal is ON.
		}

		// User Output 1 and 2 signal is OFF.
		nRtn = FAS_SetIOOutput(iBdID, 0, SERVO_OUT_BITMASK_USEROUT1 |
SERVO_OUT_BITMASK_USEROUT2);
		_ASSERT(nRtn == FMM_OK);

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

See Also

FAS_GetIOInput

FAS_GetIOInput

Read the I/O input values. For more information, refer to '1-2. Structure of Frame Type'

Syntax

```
int FAS_GetIOInput(
    int iBdID,
    DWORD* dwIOInput
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

dwIOInput

Parameter pointer which input values will be saved.

Return Value

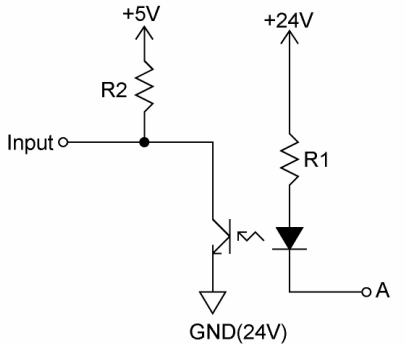
FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Ezi-SERVOⅡ Plus-E ALL has 6 inputs in total, among which 3 inputs can be customized. This function is to read the state of input port in 32bit unit and all are photocoupler isolated. (See picture)



If the voltage at point A is 24V from the external input, the input is recognized as 5V (high).

Example

Refer to FAS_SetIOInput library.

See Also

FAS_SetIOInput

FAS_SetIOOutput

Set the I/O output values. For more information, refer to '1-2. Structure of Frame Type'.

Syntax

```
int FAS_SetIOOutput(
    int iBdID,
    DWORD dwIOSetMask,
    DWORD dwIOCLRMask
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

dwIOSetMask

Bitmask value of Output when Set(ON).

dwIOCLRMask

Bitmask value of Output when Clear(OFF).

Return Value

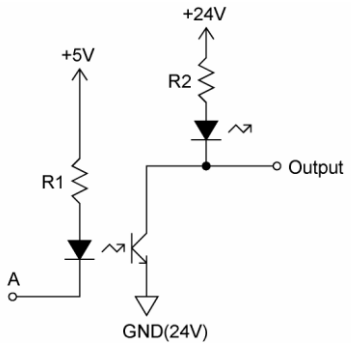
FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Ezi-SERVOⅡ Plus-E ALL has 2 outputs in total, and one of these outputs can be customized.



If output data is 1, 0V is applied to A terminal. If it is 0, + 5V is applied.

Be careful not to duplicate the bit values of dwIOSetMask and dwIOCLRMask.

Example

Refer to FAS_SetIOInput library.

See Also

FAS_GetIOOutput

FAS_GetIOOutput

Read the I/O Output. For more information, refer to '1-2. Structure of Frame Type'.

Syntax

```
int FAS_GetIOOutput(
    int iBdID,
    DWORD* dwIOOutput
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

dwIOInput

Parameter pointer which the output value will be saved.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

Refer to FAS_SetIOInput library.

See Also

FAS_SetIOOutput

FAS_GetIOAssignMap

Read I/O Assign Map. For more information, refer to '1-2. Structure of Frame Type'.

Syntax

```
int FAS_GetIOAssignMap(
    int iBdID,
    BYTE iIOPinNo,
    BYTE* nIOLogic,
    BYTE* bLevel
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

iIOPinNo

I/O Pin number to be read.

nIOLogic

Parameter pointer to save the logic value assigned to the pin.

bLevel

Parameter pointer to save Active Level value of relevant Logic.

Return Value

FMM_OK : The command ran successfully.
FMM_NOT_OPEN : The drive it not connected yet.
FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

For nIOLogic, refer to 'Motion_define.h'.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcIOAssign()
{

        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0;          //   Drive number
        BYTE iPinNo;
        DWORD dwLogicMask;
        BYTE bLevel;
        BYTE i;
        int nRtn;
```

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

	// Check the input pin assignment information.
	for (i=0; i</*Input Pin Count*/12; i++)
	{
		nRtn = FAS_GetIOAssignMap(iBdID, i, &dwLogicMask, &bLevel);
		_ASSERT(nRtn == FMM_OK);

		if (dwLogicMask != IN_LOGIC_NONE)
			printf("Input Pin %d : Logic Mask 0x%08X (%s)₩n", i,
dwLogicMask, ((bLevel == LEVEL_LOW_ACTIVE) ? "Low Active" : "High Active"));
		else
			printf("Input Pin %d : Not assigned₩n", i);
	}

	// Assign SERVO ON Logic (Low Active) to Input pin 3.
	iPinNo = 3;		// Values 0 to 11 are possible (note: 0 to 2 are fixed).
	nRtn = FAS_SetIOAssignMap(iBdID, iPinNo, SERVO_IN_BITMASK_SERVOON,
LEVEL_LOW_ACTIVE);
	_ASSERT(nRtn == FMM_OK);

	// Check output pin assignment information.
	for (i=0; i<10/*Output Pin Count*/; i++)
	{
		nRtn = FAS_GetIOAssignMap(iBdID, 12/*Input Pin Count*/ + i,
&dwLogicMask, &bLevel);
		_ASSERT(nRtn == FMM_OK);

		if (dwLogicMask != OUT_LOGIC_NONE)
			printf("Output Pin %d : Logic Mask 0x%08X (%s)₩n", i,
dwLogicMask, ((bLevel == LEVEL_LOW_ACTIVE) ? "Low Active" : "High Active"));
		else
			printf("Output Pin %d : Not assigned₩n", i);
	}

	// Assign ALARM Logic (High Active) to output pin 9..
	iPinNo = 9;		// 0 ~ 9 value can be set (Note: 0 is fixed to COMPOUT).
	nRtn = FAS_SetIOAssignMap(iBdID, 12/*Input Pin Count*/ + iPinNo,
SERVO_OUT_BITMASK_ALARM, LEVEL_HIGH_ACTIVE);
```

```
        _ASSERT(nRtn == FMM_OK);

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

See Also

FAS_SetIOAssignMap

FAS_SetIOAssignMap

Set the I/O Assign Map. For more information, refer to '1-2. Structure of Frame Type'.

Syntax

```
int FAS_SetIOAssignMap(
    int iBdID,
    BYTE iIOPinNo,
    BYTE nLogicNo,
    BYTE bLevel
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

iIOPinNo

I / O pin number to read.

nIOLogic

Logic value to assign to the corresponding pin.

bLevel

Active Level value of the corresponding logic.

Return Value

FMM_OK : The command ran successfully.
FMM_NOT_OPEN : The drive it not connected yet.
FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.
FMM_INVALID_PARAMETER_NUM : The specified iIOPinNo or nIOLogic value is out of range.

Remarks

In order to save the current setting value in the ROM memory, FAS_SaveAllParameters 'library should be executed.

Example

Refer to FAS_GSetIOAssignMap.

See Also

FAS_GetIOAssignMap

FAS_IOAssignMapReadROM

The I / O setting status and signal level values saved in the current ROM area are read.

Syntax

```
int FAS_PosTableReadROM(

    int iBdID
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

Return Value

FMM_OK : The command ran successfully.
FMM_NOT_OPEN : The drive it not connected yet.
FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdlD does not exist.
FMC_POSTABLE_ERROR : An error occurred while reading the position table.

Remarks

Example

See Also

FAS_ GetIOAssignMap

2 - 7 . Position Control Function

Function Name	Description
FAS_SetCommandPos	Set the command position value.
FAS_SetActualPos	Set the actual position value.
FAS_GetCommandPos	Read the command position value.
FAS_GetActualPos	Read the actual position value.
FAS_GetPosError	Read the difference between the actual position and the command position.
FAS_GetActualVel	Read the actual running speed value.
FAS_ClearPosition	Set the command position and the actual position to '0'.

FAS_SetCommandPos

Set the command position value.

Syntax

```
int FAS_SetCommandPos(
    int iBdID,
    long lCmdPos
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lCmdPos

Command Position value to be set.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdlD does not exist.

Remarks

The user sets the command position (pulse output counter) value.

It is mainly used to set the current position to the desired coordinate value.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcClearPosition()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0;          //   Drive number
        int nRtn;

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

        // Initialize Command Position, Actual Position to 0.
```

```
        nRtn = FAS_SetCommandPos(iBdID, 0);
        _ASSERT(nRtn == FMM_OK);
        nRtn = FAS_SetActualPos(iBdID, 0);
        _ASSERT(nRtn == FMM_OK);

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

See Also

FAS_SetActualPos

FAS_SetActualPos

Set the Actual Position value of the motor.

Syntax

```
int FAS_SetActualPos(
    int iBdID,
    long lActPos
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lActPos

Actual position value to be set.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Set the encoder feedback counter value to the value the user wants.

Example

Refer to FAS_GetActualPos library.

See Also

FAS_SetCommandPos

FAS_GetCommandPos

Read the command position value of the motor.

Syntax

```
int FAS_GetCommandPos(
    int iBdID,
    long* lCmdPos
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lCmdPos

Parameter pointer where command position value is stored.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdlD does not exist.

Remarks

Read the command position (pulse output counter) value.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcDisplayStatus()
{

        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0;          //   Drive number
        long lValue;
        int nRtn;

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

        // Check the position information of Ezi-SERVOⅡ.
        nRtn = FAS_GetCommandPos(iBdID, &lValue);
```

```
        _ASSERT(nRtn == FMM_OK);
        printf("CMDPOS : %d ₩n", lValue);
        nRtn = FAS_GetActualPos(iBdID, &lValue);
        _ASSERT(nRtn == FMM_OK);
        printf("ACTPOS : %d ₩n", lValue);
        nRtn = FAS_GetPosError(iBdID, &lValue);
        _ASSERT(nRtn == FMM_OK);
        printf("POSERR : %d ₩n", lValue);
        nRtn = FAS_GetActualVel(iBdID, &lValue);
        _ASSERT(nRtn == FMM_OK);
        printf("ACTVEL : %d ₩n", lValue);

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

See Also

FAS_GetActualPos

FAS_GetActualPos

Read the actual position value of the motor.

Syntax

```
int FAS_GetActualPos(
    int iBdID,
    long* lActPos
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lActPos

Parameter pointer where actual position value is stored.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdlD does not exist.

Remarks

It is mainly used to confirm the actual position after completion of positioning.

Example

Refer to FAS_GetCommandPosition library.

See Also

FAS_GetCommandPos

FAS_GetPosError

Read the difference between the actual position and the command position.

Syntax

```
int FAS_GetPosError(
    int iBdID,
    long* lPosErr
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lPosErr

Parameter pointer where position error value is stored.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdlD does not exist.

Remarks

Example

Refer to FAS_GetCOmmandPosition library.

See Also

FAS_GetCommandPos,

FAS_GetActualPos

FAS_GetActualVel

Read the actual running speed value of the motor.

Syntax

```
int FAS_GetActualVel(
    int iBdID,
    long* lActVel
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lActVel

Parameter pointer where actual velocity value is stored.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdlD does not exist.

Remarks

Example

Refer to FAS_GetCOmmandPosition library.

See Also

FAS_ClearPosition

Set the command position and the actual position to '0' of the motor.

Syntax

```
int FAS_ClearPosition(
    int iBdID
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

Return Value

FMM_OK : The command ran successfully.
FMM_NOT_OPEN : The drive it not connected yet.
FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

The position value is set by the user.
Mainly used for system initialization.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcClearPosition()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0;          //   Drive number
        int nRtn;

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

        // Command Position, Actual Position value is initialized to 0.
        nRtn = FAS_ClearPosition(iBdID);
        _ASSERT(nRtn == FMM_OK);

        // Disconnect.
```

```
        FAS_Close(iBdID);
    }
```

See Also

FAS_SetActualPos, FAS_SetCommandPos

2 - 8 . Drive Status Control Function

Function Name	Description
FAS_GetIOAxisStatus	Read control I / O status and operation status flag value. : Returns current input status value, output setting status value and operation status Flag value.
FAS_GetMotionStatus	It reads the current operation progress and operating PT number. : Returns the command position, actual position, speed value, etc.
FAS_GetAllStatus	Read all at once, including the current state. : This is a combination of 'FAS_GetIOAxisStatus' and 'FAS_ GetMotionStatus'.
FAS_GetAxisStatus	Reads the operation status flag of the drive.

FAS_GetIOAxisStatus

Read both I / O value and Motor Axis Status value of the relevant drive.

Syntax

```
int FAS_GetIOAxisStatus(
    int iBdID,
    DWORD* dwInStatus,
    DWORD* dwOutStatus,
    DWORD* dwAxisStatus
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

dwInStatus

Parameter pointer where I/O Input value is stored.

dwOutStatus

Parameter pointer where I/O Output value is stored.

dwAxisStatus

Parameter pointer where Axis Status value of the corresponding motor is stored.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

Refer to FAS_MoveSingleAxisAbsPos library.

See Also

FAS_GetMotionStatus

It reads the current operation progress and operating PT number.

Syntax

```
int FAS_GetMotionStatus(
    int iBdID,
    long* lCmdPos,
    long* lActPos,
    long* lPosErr,
    long* lActVel,
    WORD* wPosItemNo
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lCmdPos

Parameter pointer where Command Position value is stored.

lActPos

Parameter pointer where Actual Position value is stored.

lPosErr

Parameter pointer where Position Error value is stored.

lActVel

Parameter pointer where Actual Velocity value is stored.

wPosItemNo

Parameter pointer where currently operating Item Number value of the Position Table is stored.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

Refer to FAS_MoveSingleAxisAbsPos library.

See Also

FAS_GetAllStatus

Reads I / O value, Axis Status, and Motion Status of the drive.

Syntax

```
int FAS_GetAllStatus(
    int iBdID,
    DWORD* dwInStatus,
    DWORD* dwOutStatus,
    DWORD* dwAxisStatus,
    long* lCmdPos,
    long* lActPos,
    long* lPosErr,
    long* lActVel,
    WORD* wPosItemNo
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

dwInStatus

Parameter pointer where I/O Input value is stored.

dwOutStatus

Parameter pointer where I/O Output value is stored.

dwAxisStatus

Parameter pointer where Axis Status value of the corresponding motor is stored.

lCmdPos

Parameter pointer where Command Position value is stored.

lActPos

Parameter pointer where Actual Position value is stored.

lPosErr

Parameter pointer where Position Error value is stored.

lActVel

Parameter pointer where Actual Velocity value is stored.

wPosItemNo

Parameter pointer where currently operating Item Number value of the Position Table is stored.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

Refer to FAS_MoveSingleAxisAbsPos library.

See Also

FAS_GetAxisStatus
FAS_GetMotionStatus

FAS_GetAxisStatus

Read the Axis Status value of the motor. For more information about Status Flag, refer to '1-2. Structure of Frame Type'.

Syntax

```
int FAS_GetAxisStatus(
    int iBdID,
    DWORD* dwAxisStatus
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

dwAxisStatus

Parameter pointer where Axis Status is stored.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

Refer to FAS_MoveSingleAxisAbsPos library.

See Also

2 - 9 . Operation Control Function

Function Name	Description
FAS_MoveStop	Stop the motor in motion with deceleration.
FAS_EmergencyStop	Stop the motor in motion immediately without deceleration.
FAS_MoveOriginSingleAxis	Start homing operation.
FAS_MoveSingleAxisAbsPos	Move as much as given absolute position value.
FAS_MoveSingleAxisIncPos	Move as much as given incremental position value.
FAS_MoveToLimit	Move to the position where limit sensor is detected.
FAS_MoveVelocity	Start driving at the given velocity and direction : Used for jog operation.
FAS_PositionAbsOverride	Change the target absolute position value [pulse] while it is in motion.
FAS_PositionIncOverride	Change the target incremental position value [pulse] while it is in motion.
FAS_VelocityOverride	Change the velocity value [pps] while it is in motion.
FAS_MoveLinearAbsPos	Linear Interpolation operation is performed by the absolute position value given to two or more drives.
FAS_MoveLinearIncPos	Linear Interpolation operation is performed as much as the incremental position value given to two or more drives.
FAS_MoveLinearAbsPos2[*1]	Improved version of **FAS_MoveLinearAbsPos.** Acceleration and Deceleration were improved.
FAS_ MoveLinearIncPos2[*1]	Improved version of **FAS_MoveLinearIncPos.** Acceleration and Deceleration were improved.
FAS_MoveSingleAxisAbsPosEx	Move as much as given absolute position value. You can set the acceleration and deceleration times.
FAS_MoveSingleAxisIncPosEx	Move as much as given incremental position value. You can set the acceleration and deceleration times.
FAS_MoveVelocityEx	Start driving at the given velocity and direction : Used for jog operation. You can set the acceleration and deceleration times.
FAS_MovePause	Pauses the operation while it is in operation or resumes the operation while in the paused state.

[*1] These functions are available from Firmware [ver.6.1.20.18] or higher version.

FAS_MoveStop

Stop the motor with deceleration.

Syntax

```
int FAS_MoveStop(
    int iBdID,
    );
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

Refer to FAS_MoveSingleAxisAbsPos library.

See Also

FAS_EmergencyStop

Stop the motor immediately without deceleration.

Syntax

```
int FAS_EmergencyStop(
    int iBdID,
    );
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

Return Value

FMM_OK : The command ran successfully.
FMM_NOT_OPEN : The drive it not connected yet.
FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdlD does not exist.

Remarks

This function does not have a deceleration step, so pay attention to the impact on the machine.

Example

Refer to FAS_MoveSingleAxisAbsPos library.

See Also

FAS_MoveOriginSingleAxis

Search the origin of the system.

For more information, refer to 「User Manual_Text 8.3 Origin Return」.

Syntax

```
int FAS_MoveOriginSingleAxis(
    int iBdID,
    );
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

Refer to FAS_MoveSingleAxisAbsPos library.

See Also

FAS_MoveSingleAxisAbsPos

Move as much as given absolute position value.

Syntax

```
int FAS_MoveSingleAxisAbsPos(
    int iBdID,
    long lAbsPos,
    DWORD lVelocity,
    );
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lAbsPos

Absolute position value of the target position.

lVelocity

Speed value of the motor.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdlD does not exist.

Remarks

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcMove()
{

        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0;          //   Drive number
        DWORD dwAxisStatus, dwInput;
        EZISERVO_AXISSTATUS stAxisStatus;
        long lAbsPos, lIncPos, lVelocity;
        int nRtn;

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

```
{
        // Connection failed.

        return;
}

// Check Error and Servo On status.
nRtn = FAS_GetAxisStatus(iBdID, &dwAxisStatus);
_ASSERT(nRtn == FMM_OK);
stAxisStatus.dwValue = dwAxisStatus;

//if (dwAxisStatus & 0x00000001)
if (stAxisStatus.FFLAG_ERRORALL)
        FAS_ServoAlarmReset(iBdID);
//if ((dwAxisStatus & 0x00100000) == 0x00)
if (stAxisStatus.FFLAG_SERVOON == 0)
        FAS_ServoEnable(iBdID, TRUE);

// Check Input status.
nRtn = FAS_GetIOInput(iBdID, &dwInput);
_ASSERT(nRtn == FMM_OK);

if (dwInput & (SERVO_IN_LOGIC_STOP | SERVO_IN_LOGIC_PAUSE |
SERVO_IN_LOGIC_ESTOP))
        FAS_SetIOInput(iBdID, 0, SERVO_IN_LOGIC_STOP | SERVO_IN_LOGIC_PAUSE |
SERVO_IN_LOGIC_ESTOP);

// Move the Motor by 15000 pulses.
lIncPos = 15000;
lVelocity = 30000;
nRtn = FAS_MoveSingleAxisIncPos(iBdID, lIncPos, lVelocity);
_ASSERT(nRtn == FMM_OK);

// Wait until the Motion command is completed.
do
{
        Sleep(1);

        nRtn = FAS_GetAxisStatus(iBdID, &dwAxisStatus);
        _ASSERT(nRtn == FMM_OK);
        stAxisStatus.dwValue = dwAxisStatus;
}
while (stAxisStatus.FFLAG_MOTIONING);

// Move the Motor to '0' absolute position.
```

```
        lAbsPos = 0;
        lVelocity = 20000;
        nRtn = FAS_MoveSingleAxisAbsPos(iBdID, lAbsPos, lVelocity);
        _ASSERT(nRtn == FMM_OK);

        // Wait until the Motion command is completed.
        do
        {
                Sleep(1);
                nRtn = FAS_GetAxisStatus(iBdID, &dwAxisStatus);
                _ASSERT(nRtn == FMM_OK);
                stAxisStatus.dwValue = dwAxisStatus;
        }
        while (stAxisStatus.FFLAG_MOTIONING);

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

See Also

FAS_MoveSingleAxisIncPos

Move as much as given incremental position value.

Syntax

```
int FAS_MoveSingleAxisIncPos(
    int iBdID,
    long lIncPos,
    DWORD lVelocity
    );
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lIncPos

Increamental position value of the target position.

lVelocity

Speed value of the motor.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

Refer to FAS_MoveSingleAxisAbsPos library.

See Also

FAS_MoveToLimit

Move to the position where limit sensor is detected.

Syntax

```
int FAS_MoveToLimit(
    int iBdID,
    DWORD lVelocity,
    int iLimitDir,
    );
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lVelocity

Speed value of the motor.

iLimitDir

Direction of Limit to go to.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

Refer to FAS_MoveSingleAxisAbsPos library.

See Also

FAS_MoveVelocity

Move the motor in the desired direction and speed. Used in jog operation.

Syntax

```
int FAS_MoveVelocity(
    int iBdID,
    DWORD lVelocity,
    int iVelDir
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lVelocity

Speed value of the motor.

iVelDir

Direction to move.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

Refer to FAS_MoveSingleAxisAbsPos library.

See Also

FAS_PositionAbsOverride

Change the target absolute position value [pulse] while it is in motion.

Syntax

```
int FAS_PositionAbsOverride(
    int iBdID,
    long lOverridePos
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lOverridePos

Absolute target position.

Return Value

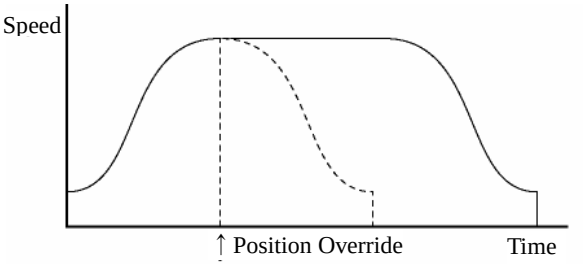
FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

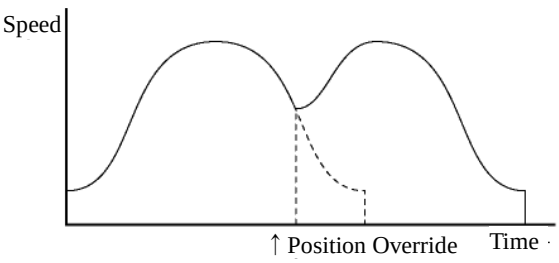
FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

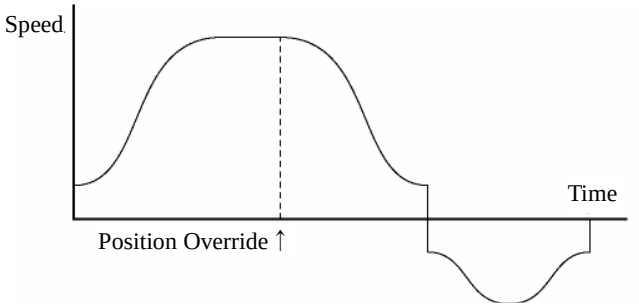
1) If the target position is set to a coordinate farther than the original target position during acceleration or constant speed, it will operate in the speed pattern up to that point and stop at the changed target position.



2) If you change the target position during deceleration, the speed pattern accelerates up to constant velocity again and stops at the changed target position.



3) If the changed target position is set closer to the original target position, it moves to the changed target position and stops.



4) It cannot be used simultaneously with the FAS_PositionAbsOverride library.
It cannot be used simultaneously with the FAS_VelocityOverride library.

Example

Refer to FAS_MoveSingleAxisAbsPos library.

See Also

FAS_PositionIncOverride

FAS_PositionIncOverride

Change the target incremental position value [pulse] while it is in motion.

Syntax

```
int FAS_PositionIncOverride(
    int iBdID,
    long lOverridePos
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lOverridePos

Target incremental position.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

1) Refer to FAS_PositionAbsOverride library.

2) It cannot be used simultaneously with the FAS_PositionIncOverride library.
It cannot be used simultaneously with the FAS_VelocityOverride library.

Example

Refer to FAS_MoveSingleAxisAbsPos library.

See Also

FAS_PositionAbsOverride

FAS_VelocityOverride

Change the velocity value [pps] while it is in motion.

Syntax

```
int FAS_VelocityOverride(
    int iBdID,
    DWORD lVelocity
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lVelocity

Target velocity value.

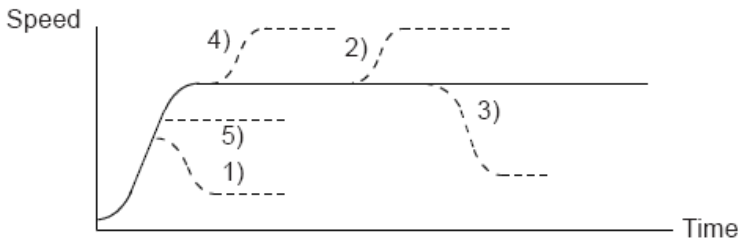
Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks



1) In case of ((change speed) <(speed before change)), 'change speed' is reached through acceleration / deceleration using new speed pattern.

5) In case of ((change speed) ≥ (speed before change)), acceleration / deceleration without speed pattern is reached to 'change speed'.

4) Until 'speed before change' is reached without changing the speed pattern, and until 'change speed' is reached with the new speed pattern characteristic.

2), 3) After acceleration / deceleration is completed, it reaches 'change speed' according to the speed pattern characteristic of 'change speed'.

- It cannot be used simultaneously with the FAS_PositionIncOverride library.
- It cannot be used simultaneously with the FAS_PositionAbsOverride library.

Example

Refer to FAS_MoveSingleAxisAbsPos library.

See Also

FAS_MoveLinearAbsPos

Move as much as given absolute position value. You can set the acceleration and deceleration times.

Syntax

```
int FAS_MoveLinearAbsPos(
    BYTE nNoOfBds,
    int* iBdID,
    long* lplAbsPos,
    DWORD lFeedreat,
    DWORD wAcceltime
);
```

Parameters

nNoOfBds

Number of drives to execute linear motion

iBdID

ID array of Drives

lplAbsPos

Movement position's arrangement of Drives

lFeedrate

Linear velocity value during movement

wAcceltime

Time value of acceleration / deceleration section during movement

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

FAS_ MoveLinearIncPos

Move as much as given incremental position value. You can set the acceleration and deceleration times.

Syntax

```
int FAS_MoveLinearIncPos(
    BYTE nNoOfBds,
    int* iBdID,
    long* lplIncPos,
    DWORD lFeedreat,
    DWORD wAcceltime
);
```

Parameters

nNoOfBds

Number of drives to execute linear motion

iBdID

ID array of Drives

lplAbsPos

Movement position's arrangement of Drives

lFeedrate

Linear velocity value during movement

wAcceltime

Time value of acceleration / deceleration section during movement

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

FAS_MoveLinearAbsPos2

Improved version of FAS_MoveLinearAbsPos.

Acceleration and Deceleration were improved.

Syntax

```
int FAS_MoveLinearAbsPos2(
    BYTE nNoOfBds,
    int* iBdID,
    long* lplAbsPos,
    DWORD lFeedreat,
    DWORD wAcceltime
);
```

Parameters

nNoOfBds

Number of drives to execute linear motion

iBdID

ID array of Drives

lplAbsPos

Movement position's arrangement of Drives

lFeedrate

Linear velocity value during movement

wAcceltime

Time value of acceleration / deceleration section during movement

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

FAS_ MoveLinearIncPos2

Improved version of FAS_MoveLinearIncPos.

Acceleration and Deceleration were improved.

Syntax

```
int FAS_MoveLinearIncPos2(
    BYTE nNoOfBds,
    int* iBdID,
    long* lplIncPos,
    DWORD lFeedreat,
    DWORD wAcceltime
);
```

Parameters

nNoOfBds

Number of drives to execute linear motion

iBdID

ID array of Drives

lplAbsPos

Movement position's arrangement of Drives

lFeedrate

Linear velocity value during movement

wAcceltime

Time value of acceleration / deceleration section during movement

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

FAS_MoveSingleAxisAbsPosEx

Move as much as given absolute position value. You can set the acceleration and deceleration times.

Syntax

```
int FAS_MoveSingleAxisAbsPosEx(
    int iBdID,
    long lAbsPos,
    DWORD lVelocity,
    MOTION_OPTION_EX* lpExOption
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lAbsPos

Target absolute position

lVelocity

Speed value of the motor.

lpExOption

Custom option.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Refer to MOTION_OPTION_EX struct.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcMoveEx()
{

        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0;          //   Drive number
        DWORD dwAxisStatus, dwInput;
        EZISERVO_AXISSTATUS stAxisStatus;
        long lAbsPos, lIncPos, lVelocity;
        MOTION_OPTION_EX opt = {0};
```

```
int nRtn;

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

	return;
}

// Move the motor with a certain acceleration and deceleration time : FAS_MoveSingleAxisIncPosEx
lIncPos = 15000;
lVelocity = 30000;

opt.flagOption.BIT_USE_CUSTOMACCEL = 1;
opt.flagOption.BIT_USE_CUSTOMDECEL = 1;

opt.wCustomAccelTime = 50;
opt.wCustomDecelTime = 200;

nRtn = FAS_MoveSingleAxisIncPosEx(iBdID, lIncPos, lVelocity, &opt);
_ASSERT(nRtn == FMM_OK);

// Wait for the Motion command to finish completely.
do
{
	Sleep(1);

	nRtn = FAS_GetAxisStatus(iBdID, &dwAxisStatus);
	_ASSERT(nRtn == FMM_OK);
	stAxisStatus.dwValue = dwAxisStatus;
}
while (stAxisStatus.FFLAG_MOTIONING);

// Move it to position 0.
lAbsPos = 0;
lVelocity = 20000;
nRtn = FAS_MoveSingleAxisAbsPos(iBdID, lAbsPos, lVelocity);
_ASSERT(nRtn == FMM_OK);

// Wait for the Motion command to finish completely.
do
{
	Sleep(1);
```

```
            nRtn = FAS_GetAxisStatus(iBdID, &dwAxisStatus);
            _ASSERT(nRtn == FMM_OK);
            stAxisStatus.dwValue = dwAxisStatus;
        }
        while (stAxisStatus.FFLAG_MOTIONING);

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

See Also

FAS_MoveSingleAxisIncPosEx

Move as much as given incremental position value. You can set the acceleration and deceleration times.

Syntax

```
int FAS_MoveSingleAxisIncPosEx(
    int iBdID,
    long lIncPos,
    DWORD lVelocity,
    MOTION_OPTION_EX* lpExOption
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lIncPos

Target incremental position.

lVelocity

Speed value of the motor.

lpExOption

Custom option.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

See Also

FAS_MoveVelocityEx

Move the motor in the desired direction, speed, and acc/dec time. Used in jog operation.

Syntax

```
int FAS_ MoveVelocityEx (
    int iBdID,
    DWORD lVelocity,
    int iVelDir,
    VELOCITY_OPTION_EX* lpExOption
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

lVelocity

Speed value of the motor.

iVelDir

Direction to move (0: -Jog, 1: +Jog)

lpExOption

Custom option.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Refer to VELOCITY_OPTION_EX struct.

Example

```
#include "FAS_ EziMOTIONPlusE.h"

void funcMoveVelocityEx()
{
        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0;          //   Drive number
        long lVelocity;
        VELOCITY_OPTION_EX opt = {0};
        int nRtn;

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

```
                    // Connection failed.

                    return;
                }

                // Move the motor with a certain acceleration and deceleration time. :
FAS_MoveSingleAxisIncPosEx
                lVelocity = 30000;

                opt.flagOption.BIT_USE_CUSTOMACCDEC = 1;
                opt.wCustomAccDecTime = 300;

                nRtn = FAS_MoveVelocityEx(iBdID, lVelocity, DIR_INC, &opt);
                _ASSERT(nRtn == FMM_OK);

                Sleep(5000);
                FAS_MoveStop(iBdID);
}
```

See Also

2 - 1 0 . Position Table Control Function

Function Name	Description
FAS_PosTableReadItem	Read item values of desired position table from RAM area.
FAS_PosTableWriteItem	Stores the item value of a desired position table in RAM area.
FAS_PosTableWriteROM	Store all position table values in ROM area : All 256 PT values are saved.
FAS_PosTableReadROM	Read position table values in ROM area : Read all 256 PT values.
FAS_PosTableRunItem	Starts operation sequentially from the designated position table.
FAS_PosTableReadOneItem	Read desired item value of desired position table from RAM area.
FAS_PosTableWriteOneItem	Store desired item value of desired position table in RAM area.

FAS_PosTableReadItem

Read desired item values of position table.

Syntax

```
int FAS_PosTableReadItem(
    int iBdID,
    WORD wItemNo,
    LPITEM_NODE lpItem
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

wItemNo

Item number to read.

lpItem

Item structure pointer to store Item value.

Return Value

FMM_OK : The command ran successfully.
FMM_NOT_OPEN : The drive it not connected yet.
FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.
FMM_INVALID_PARAMETER_NUM : wItemNo is out of range.

Remarks

Example

```
#include "FAS_EziMOTIONPlusE.h"

void funcPosTable()
{

        BYTE sb1 = 192, sb2 = 168, sb3=0, sb4=2 // IP :192.168.0.2
        int iBdID = 0;          //   Drive number
        WORD wItemNo;
        ITEM_NODE nodeItem;
        int nRtn;

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

```
            // Connection failed.

            return;
        }

        // Read position table value of item No. 20 and change the position value.
        wItemNo = 20;
        nRtn = FAS_PosTableReadItem(iBdID, wItemNo, &nodeItem);
        _ASSERT(nRtn == FMM_OK);

        nodeItem.lPosition = 260000;  // Change the position value to 260000.
        nodeItem.wBranch = 23;                  // Set next command to No. 23.
        nodeItem.wContinuous = 1;               // Set the next command to continue
immediately without deceleration.

        nRtn = FAS_PosTableWriteItem(iBdID, wItemNo, &nodeItem);
        _ASSERT(nRtn == FMM_OK);

        // The ROM value is loaded, ignoring the currently modified position table data.
        nRtn = FAS_PosTableReadROM(iBdID);
        _ASSERT(nRtn == FMM_OK);

        // Save current modified position table data to ROM.
        nRtn = FAS_PosTableWriteROM(iBdID);
        _ASSERT(nRtn == FMM_OK);

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

See Also

FAS_PosTableWriteItem

FAS_PosTableWriteItem

Modify the desired item in the position table.

Syntax

```
int FAS_PoslableWriteItem(
    int iBdID,
    WORD wItemNo,
    LPITEM_NODE lpItem
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

wItemNo

Item number to change

lpItem

Item structure pointer to modify.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

FMC_POSTABLE_ERROR : An error occurred while writing the position table.

FMM_INVALID_PARAMETER_NUM : wItemNo is out of range.

Remarks

The area where position table data is stored is RAM and ROM.

This function is to save data in RAM area. When power is off, data is deleted.

Example

See Also

FAS_PosTableWriteROM

Saves all current Position Table Items to ROM area.

Syntax

```
int FAS_PosTableWriteROM(
    int iBdID
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

Return Value

FMM_OK : The command ran successfully.
FMM_NOT_OPEN : The drive it not connected yet.
FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.
FMC_POSTABLE_ERROR : An error occurred while saving the position table.

Remarks

The area where position table data is stored is RAM and ROM.
This function saves the data in the ROM area. The data is saved even when the power is turned off.

Example

See Also

FAS_PosTableReadROM

FAS_PosTableReadROM

Reat the Position Table Item values currently saved in the ROM area.

Syntax

```
int FAS_PosTableReadROM(
    int iBdID
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

FMC_POSTABLE_ERROR : An error occurred while reading the position table.

Remarks

Example

See Also

FAS_PosTableWriteROM

FAS_PosTableRunItem

Starts operation sequentially from the designated position table.

Syntax

```
int FAS_PosTableRunItem(
    int iBdID,
    WORD wItemNo
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

wItemNo

Item number to start operation.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

FMM_INVALID_PARAMETER_NUM : wItemNo is out of range.

Remarks

Example

See Also

FAS_GetAllStatus

FAS_MoveStop

FAS_EmergencyStop

FAS_PosTableReadOneItem

Read desired item value of position table.

Syntax

```
int FAS_PosTableReadOneItem(
    int iBdID,
    WORD wItemNo,
    WORD wOffset,
    long* lPosItemVal
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

wItemNo

Item number to read.

wOffset

Offset value of the item to read. (Refer to 「1-2-6. Position Table Item」)

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

FMM_INVALID_PARAMETER_NUM : wItemNo is out of range.

Remarks

Example

See Also

FAS_PosTableReadItem

FAS_PosTableWriteOneItem

FAS_PosTableWriteOneItem

Modify the desired Item value in the Position Table.

Syntax

```
int FAS_PoslableWriteOneItem(
int iBdID,
WORD wItemNo,
WORD wOffset,
long lPosItemVal
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

wItemNo

Item number to change

wOffset

Offset value of the item to read. (Refer to 「1-2-6. Position Table Item」)

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdlD does not exist.

FMC_POSTABLE_ERROR : An error occurred while writing the position table.

FMM_INVALID_PARAMETER_NUM : wItemNo is out of range.

Remarks

Example

See Also

FAS_PosTableWriteItem

FAS_PosTableReadOneItem

2 - 1 1 . Other Control Function

Function Name	Description
FAS_TriggerOutput_RunA	Function to generate output signal at desired position
FAS_TriggerOutput_Status	Function to check whether output signal (COMP) is generated
FAS_MovePush	Function to move a specified torque from a certain position
FAS_GetPushStatus	Function to check the current push motion state

FAS_TriggerOutput_RunA

Generates / ends the digital output signal (COMP pin) at a specific position during operation by position command.

Syntax

```
int FAS_TriggerOutput_RunA(
int iBdID,
BOOL bStartTrigger,
long lStartPos,
DWORD dwPeriod,
DWORD dwPulseTime,
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

bStartTrigger

Start / Stop command (1: Start, 0: End)

long lStartPos

Output start position [pulse]

DWORD dwPeriod

Period of output signal [pulse]

DWORD dwPulseTime

Pulse width of output signal [msec]

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdlD does not exist.

Remarks

Example

See Also

FAS_TriggerOutput_Status

FAS_ TriggerOutput_Status

Command to check whether the signal output function is working or not.

Syntax

```
int FAS_TrggerOutput_Status(
int iBdID,
BYTE* bTriggerStatus
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

bTriggerStatus

Current signal output status.

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdlD does not exist.

Remarks

Example

See Also

FAS_ TriggerOutput_RunA

FAS_MovePush

It is a function that keeps a fixed torque from a certain position during movement by position command, and stops movement when it comes into contact with work during movement, but keeps the torque.

Syntax

```
int FAS_MovePush(
int iBdID,
DWORD dwStartSpd,
DWORD dwMoveSpd,
long lPosition,
WORD wAccel,   WORD wDecel,
WORD wPushRate,
DWORD dwPushSpd,
long lEndPosition,
WORD wPushMode
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

DWORD dwStartSpd

Start speed of position command.

DWORD dwMoveSpd

Movement speed of position command.

long lPosition

Target absolute position value of position command.

WORD wAccel

Acceleration time of position command.

WORD wDecel

Deceleration time of position command.

WORD wPushRate

Torque ratio of Push motion.

DWORD dwPushSpd

Movement speed of Push motion.

long lEndPosition

Target absolute position value of Push motion.

WORD wPushMode

Mode selection of Push motion

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdlD does not exist.

Remarks

Example

See Also

FAS_GetPushStatus

FAS_GetPushStatus

Function to check the progress of the current push motion.

Syntax

```
int FAS_MovePush(
int iBdID,
BYTE* nPushStatus
);
```

Parameters

iBdID

ID number of the drive. IBdID set by the FAS_Connect function.

BYTE nPushStatus*

Push motion status value to be read. (Refer to 「1-2-1. FrameType and Data Configuration」)

Return Value

FMM_OK : The command ran successfully.

FMM_NOT_OPEN : The drive it not connected yet.

FMM_INVALID_SLAVE_NUM : The drive of corresponding iBdID does not exist.

Remarks

Example

See Also

FAS_Move Push

3. Appendix – Configuring Network Information with DHCP

3 - 1 . DHCP function

1) What is DHCP(Dynamic Host Configuration Protocol)?

→ This is a standard network protocol used to dynamically set network information for performing TCP / IP communication such as IP address.

(Network information : Gateway, Subnet, IP address)

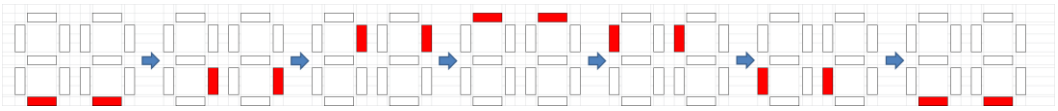
2) When not using DHCP

→ If you do not use the default gateway, subnet, and IP address of the DRIVE, you need to change and save the settings using the GUI and know the current network information.

→ When using DHCP, the Gateway, Subnet, and IP addresses are set automatically within the product. However, it is necessary to save the network information set automatically using the GUI.

3 - 2 . Network configuration using DHCP(Plus-E series)

1) Set the IP setting switch (SW1, SW2) to F, F
2) Ethernet connection to Ethernet IN Connector
3) Power on
4) 7-segments blink as below



5) When the network information is set, the IP address is displayed on the 7-segments. (After displaying aaaa.bbb.ccc.ddd, Hex. value corresponding to ddd)
6) Power off after accessing GUI and saving network information (Using Config Slave ID / IP Address)
7) Set the value of the IP configuration switchs(SW1, SW2) so that they do not overlap with the gateway from 1 to 254.
8) Power On (Setup Complete)

- DHCP dynamically configures network information, so each time you apply power, ddd can be changed in IP address(aaa.bbb.ccc.ddd). Therefore, after setting Network information by DHCP method, you must perform 6).

- **Network information can be set using DHCP only when using a PC or a router with a DHCP server function.**

FASTECH

Fast, Accurate, Smooth Motion

FASTECH Co., Ltd.

Rm#1202, 401-dong, Bucheon Techno-Park,
655, Pyeongcheon-ro, Bucheon-si Gyeonggi-do,
Republic of Korea (Zip:14502)
TEL : +82-32-234-6300 FAX : +82-32-234-6302
E-mail : fastech@fastech.co.kr
Homepage : www.fastech.co.kr