User Manual
Communication Function_Ver6

(Rev.08.05.029)
# Table of Contents

1. Communication Protocols

   1-1. Communication Functions
        1-1-1. Communication Specifications
        1-1-2. RS-485 Communication Protocol (Ver6)
        1-1-3. CRC Calculation Example
        1-1-4. Response Frame Structure and Communication Error (Ver6)

   1-2. Structure of Frame type (Ver6)
        1-2-1. Frame type and Data Configuration
        1-2-2. Parameter Lists
        1-2-3. Setup bit of Output pin
        1-2-4. Setup bit of Input pin
        1-2-5. Bit setup of Status Flag
        1-2-6. Position Table Item
        1-2-7. Information of Motors

   1-3. Program Method

2. Library for PC Program

   2-1. Library Configuration

   2-2. Communication Status Window

   2-3. Drive Link Function
        FAS_Connect
        FAS_Close
        FAS_GetSlaveInfo
        FAS_GetMotorInfo
2-4. Parameter Control Function ................................................................. 44
    FAS_SaveAllParameters ......................................................................... 45
    FAS_SetParameter .............................................................................. 47
    FAS_GetParameter .............................................................................. 48
    FAS_GetROMParameter ......................................................................... 49

2-5. Servo Control Function ...................................................................... 50
    FAS.StepAlarmReset ........................................................................... 51

2-6. Control I/O Function .......................................................................... 52
    FAS_setIOInput .................................................................................. 53
    FAS_getIOInput .................................................................................. 55
    FAS_setIOoutput ................................................................................ 56
    FAS_getIOoutput ................................................................................ 57
    FAS_getIOAssignMap .......................................................................... 58
    FAS_setIOAssignMap .......................................................................... 60
    FAS_IOAssignMapReadROM .................................................................. 61

2-7. Position Control Function ................................................................. 62
    FAS.SetCommandPos ........................................................................... 63
    FAS.SetActualPos .............................................................................. 65
    FAS.GetCommandPos .......................................................................... 66
    FAS.GetActualPos .............................................................................. 68
    FAS.GetPosError ................................................................................ 69
    FAS.GetActualVel .............................................................................. 70
    FAS.ClearPosition .............................................................................. 71

2-8. Drive Status Control Function ............................................................ 72
    FAS_GetIOAxisStatus .......................................................................... 73
    FAS_GetMotionStatus .......................................................................... 74
2—9. Running Control Function ................................................. 77
   FAS_MoveStop........................................................................... 78
   FAS_EmergencyStop................................................................. 79
   FAS_MoveOriginSingleAxis..................................................... 80
   FAS_MoveSingleAxisAbsPos..................................................... 81
   FAS_MoveSingleAxisIncPos..................................................... 83
   FAS_MoveToLimit...................................................................... 84
   FAS_MoveVelocity................................................................. 85
   FAS_PositionAbsOverride..................................................... 86
   FAS_PositionIncOverride..................................................... 88
   FAS_VelocityOverride........................................................... 89
   FAS_AllMoveStop...................................................................... 90
   FAS_AllEmergencyStop.......................................................... 91
   FAS_AllMoveOriginSingleAxis............................................... 92
   FAS_AllMoveSingleAxisAbsPos............................................... 93
   FAS_AllMoveSingleAxisIncPos............................................... 94
   FAS_MoveSingleAxisAbsPosEx................................................ 95
   FAS_MoveSingleAxisIncPosEx................................................ 97
   FAS_MoveVelocityEx............................................................. 98

2—10. Position Table Control Function ........................................ 100
   FAS_PosTableReadItem.......................................................... 101
   FAS_PosTableWriteItem....................................................... 103
   FAS_PosTableWriteROM....................................................... 104
   FAS_PosTableReadROM....................................................... 105
   FAS_PosTableRunItem.......................................................... 106
   FAS_PosTableReadOneItem................................................... 107
   FAS_PosTableWriteOneItem................................................... 108
2-11. Other Control Function ............................................................................. 109

FAS_TriggerOutput_RunA .............................................................................. 110

FAS_TriggerOutput_Status .................................................................

3. Protocol for PLC Program..................................................................................... 112
1. Communication Protocols

1-1. Communication Functions

Ezi-STEP Plus-R can control up to 16 axis by Daisy-Chain link at RS-485(two-wire).

<table>
<thead>
<tr>
<th>Specification</th>
<th>RS-485</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Type</td>
<td>Asynchronous</td>
</tr>
<tr>
<td></td>
<td>Half-duplex</td>
</tr>
<tr>
<td>Baudrate [bps]</td>
<td>19200, 38400, 57600, 115200,</td>
</tr>
<tr>
<td></td>
<td>230400, 460800, 921600</td>
</tr>
<tr>
<td>Data Type</td>
<td>8bit ASCII Code, HEX</td>
</tr>
<tr>
<td>Parity</td>
<td>No</td>
</tr>
<tr>
<td>Stop Bit</td>
<td>1bit</td>
</tr>
<tr>
<td>CRC Check</td>
<td>Yes</td>
</tr>
<tr>
<td>Max Cabling Length (Converter ↔ Drive)</td>
<td>30 m</td>
</tr>
<tr>
<td>Min Cable length between drive</td>
<td>More than 60 cm</td>
</tr>
<tr>
<td>Number of Connected Axis</td>
<td>16 axis (No. 0-F)</td>
</tr>
</tbody>
</table>

1-1-2. RS-485 Communication Protocol (Ver6)

There are 2 kinds of program version for STEP Plus-R. This manual support for Version 6 level.

<table>
<thead>
<tr>
<th>Type</th>
<th>Firmware version</th>
<th>compatibility</th>
<th>User Program(GUI) version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Level 6 (V06.0x.0xx.xx)</td>
<td>&lt;=&gt;</td>
<td>Level 6 (6.xx.x.xxx)</td>
</tr>
<tr>
<td>2</td>
<td>Level 8 (V08.xx.0xx.xx)</td>
<td>&lt;=&gt;</td>
<td>Level 8 (8.xx.x.xxx)</td>
</tr>
</tbody>
</table>

After connect the User Program(GUI). Version number can be check in ‘About Plus-R GUI…’ menu in ‘Help’ menu.
1) Overview of communication FRAME

PC Communication Port \#n Sending Frame Ezi-STEP Plus-R Response Frame

2) Basic structure of Frame

<table>
<thead>
<tr>
<th>Header</th>
<th>Frame Data</th>
<th>Tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xAA 0xCC</td>
<td>5-252 bytes</td>
<td>0xAA 0xEE</td>
</tr>
</tbody>
</table>

① 0xAA: Delimited byte
② 0xAA 0xCC: Indicate header of the frame.
③ 0xAA 0xEE: Indicate tail of the frame.
④ If any of the Frame data is ‘0xAA’, ‘0xAA’ should be added right after it (byte stuffing)
⑤ If any data following ‘0xAA’ is not ‘0xAA’, ‘0xCC’ or ‘0xEE’, it indicates an error.

Detailed Frame Data is configured as follows:

<table>
<thead>
<tr>
<th>Slave ID</th>
<th>Frame type</th>
<th>Data</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>1 byte</td>
<td>0-248 bytes</td>
<td>2 bytes</td>
</tr>
</tbody>
</table>

① Slave ID: Divide module number (0-15) connected to the PC communication port.
② Frame type: Designate command type of relevant frames. For the command type, refer to ‘Frame Type and Data Configuration’ section.
③ Data: Data structure and length is set according to Frame type. For more information, refer to ‘Frame Type and Data Configuration’ section.
④ CRC: To check an error which occurs during communication. ‘0xA001’ of a polynomial factor in CRC(Cyclic Redundancy Check), ‘X16+X15+X2+1’ of a polynomial factor in CRC-16-IBM (Cyclic Redundancy Check) is used. CRC calculation is performed for all items (Slave ID, Frame type, Data) prior to CRC item.

1-1-3. CRC Calculation Example

The following program source is included in a file (file name: CRC_Checksum.c) provided with the product.

1) ’0xA001’ of CRC16

```c
const unsigned short TABLE_CRCVALUE[] =
{
  0x0000, 0x00C1, 0x01B1, 0x0140, 0x0301, 0x03C0, 0x02B0, 0x0241,
  0x0601, 0x06C0, 0x07B0, 0x0741, 0x0500, 0x05C1, 0x04B1, 0x0440,
  0x0CB1, 0x0CD0, 0x0D41, 0x00F0, 0x0FC1, 0x0EB1, 0x0E40,
  0x0A00, 0x0AC1, 0x0CB1, 0x0B40, 0x09C0, 0x0900, 0x0B80, 0x0B41,
  0x0B01, 0x0B8C, 0x0B00, 0x0D91, 0x01B0, 0x01D0, 0x00B1, 0x01A0,
  0x1000, 0x10EC1, 0x10F81, 0x11F0, 0x10D0, 0x10C0, 0x10D4, 0x1040,
  0x1400, 0x14D41, 0x1558, 0x1540, 0x1701, 0x1760, 0x16B0, 0x1641,
  0x1020, 0x12D0, 0x13B0, 0x1341, 0x1100, 0x101C1, 0x10B1, 0x1040,
  0x10F0, 0x13D0, 0x13B0, 0x14F1, 0x1330, 0x13C1, 0x12F81, 0x1240,
  0x1360, 0x13F81, 0xe740, 0xe750, 0xe5F0, 0xe5C0, 0xe6B0, 0xe641,
  0xe3C0, 0xe3C1, 0xe3FD1, 0xe3D40, 0xe3F0, 0xe3F0, 0xe3D80, 0xe3F41,
  0xe3A01, 0xe3A0, 0xe3B0, 0xe3B41, 0xe390, 0xe3F1, 0xe3F81, 0xe380,
  0xe380, 0xe38C1, 0xe38B1, 0xe3940, 0xe3D0, 0xe3C0, 0xe3A0, 0xe3A41,
  0xe3E01, 0xe3E0, 0xe3F80, 0xe3F41, 0xe2D0, 0xe2D1, 0xe2C1, 0xe2B0, 0xe2C40,
};
```
unsigned short CalcCRC(unsigned char* pDataBuffer, unsigned long usDataLen)
{
    unsigned char nTemp;
    unsigned short wCRCWord = 0xFFFF;

    while (usDataLen--)
    {
        nTemp = wCRCWord ^ *(pDataBuffer++);
        wCRCWord >>= 8;
        wCRCWord ^= TABLE_CRCVALUE[nTemp];
    }
    return wCRCWord;
}

2) 'X16+X15+X2+1' of CRC-16-IBM

unsigned short CalcCRCbyAlgorithm(unsigned char* pDataBuffer, unsigned long usDataLen)
{
    const unsigned short POLYNOMIAL = 0xA001;
    unsigned short wCrc;
    int iByte, iBit;

    /* Initialize CRC */
    wCrc = 0xffff;

    for (iByte = 0; iByte < usDataLen; iByte++)
    {
        /* Exclusive-OR the byte with the CRC */
        wCrc ^= *(pDataBuffer + iByte);

        /* Loop through all 8 data bits */
        for (iBit = 0; iBit < 8; iBit++)
        {
            if (wCrc & (1 << iBit))
            {
                wCrc = (wCrc << 1) ^ POLYNOMIAL;
            }
            else
            {
                wCrc <<= 1;
            }
        }
    }
    return wCrc;
}
for (iBit = 0; iBit <= 7; iBit++)
{
    /* If the LSB is 1, shift the CRC and XOR the polynomial mask with the CRC */

    // Note - the bit test is performed before the rotation, so can't move the << here
    if (wCrc & 0x0001)
    {
        wCrc >>= 1;
        wCrc ^= POLYNOMIAL;
    }
    else
    {
        // Just rotate it
        wCrc >>= 1;
    }
}
return wCrc;
1-1-4. Response Frame Structure and Communication Error (Ver6)

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

<table>
<thead>
<tr>
<th>Slave ID</th>
<th>Frame Type</th>
<th>Data</th>
<th>CRC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 byte</td>
<td>1 byte</td>
<td>1 byte</td>
<td>0-247 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status</td>
<td>Response data</td>
</tr>
</tbody>
</table>

① Slave ID: Same to sending Frame.
    (When this is not same to sending data, need to recognize as the error status.)
② Frame type: Same to sending Frame.
    (When this is not same to sending data, need to recognize as the error status.)
③ Data: When simple executive instructions are sent, this data cannot be read. However, in case of response, 1 byte is included to the display of communication status (error / normal status).

The code by bytes means the ‘Communication status’ as follows.

<table>
<thead>
<tr>
<th>Hexa Code</th>
<th>Decimal Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00</td>
<td>0</td>
<td>Communication is normal.</td>
</tr>
<tr>
<td>0xB0</td>
<td>128</td>
<td>Frame Type Error: Responded Frame type cannot be recognized.</td>
</tr>
<tr>
<td>0xB1</td>
<td>129</td>
<td>Data error, ROM data read/write error: Responded data value is aside from the given range.</td>
</tr>
<tr>
<td>0xB2</td>
<td>130</td>
<td>Received Frame Error: Frame data received is out of this specification.</td>
</tr>
<tr>
<td>0xB5</td>
<td>133</td>
<td>Running Command Failure: The user has tried to execute new running commands in wrong condition as follows.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1) currently motor is running</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) currently motor is stopping</td>
</tr>
<tr>
<td></td>
<td></td>
<td>3) Servo is OFF status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4) try to Z-pulse Origin without external encoder</td>
</tr>
<tr>
<td>0xB6</td>
<td>134</td>
<td>RESET Failure: The user has tried to execute new running commands in wrong condition as follows.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1) STEP is ON status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) Already reset status by external input signal</td>
</tr>
<tr>
<td>0xAA</td>
<td>170</td>
<td>CRC Error: When received frame data is error by external noise, sending side of DLL Library is automatically trying to send 1 more time of communication signal.</td>
</tr>
</tbody>
</table>

Caution

1) If ‘Header’ and ‘Slave ID’ values in the sending 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 type (Ver6)

### 1-2-1. Frame type and Data Configuration

(1) The following table explains the content and configuration by frame type of data.

<table>
<thead>
<tr>
<th>Frame Type</th>
<th>Library Name</th>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x01 (1)</td>
<td>FAS_GetSlaveInfo</td>
<td>Connected slave type and program version information are required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 0 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 1-248 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-246 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Slave type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASCII string with NULL byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( strlen() + 1 bytes)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>♦ Slave type: 20 : Ezi-STEP Plus-R ST</td>
</tr>
<tr>
<td></td>
<td></td>
<td>60 : Ezi-STEP Plus-R MINI</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 : Ezi-SERVO Plus-R ST</td>
</tr>
<tr>
<td>0x05 (5)</td>
<td>FAS_GetMotorInfo</td>
<td>Connected motor type and manufacturer information are required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 0 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 1-248 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0-246 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Motor type</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ASCII string with NULL byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>( strlen() + 1 bytes)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>♦ Motor type: refer to 「1-1-5.Information of Motors」</td>
</tr>
<tr>
<td>0x10 (16)</td>
<td>FAS_SaveAllParameters</td>
<td>Save currently set parameters &amp; assigned signals in the ROM of the drive. Even the drive is powered off, saving these data &amp; parameters are possible. Values set at 'FAS_SetParameter' &amp; 'FAS_SetIOAssignMap' are saved together.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 0 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status</td>
</tr>
<tr>
<td>0x11 (17)</td>
<td>FAS_GetROMParameter</td>
<td>Specific parameter values in the ROM are recognized.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parameter number (0-28)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 5 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parameter value</td>
</tr>
</tbody>
</table>

Refer to 「1-2-2.Parameter List」
<table>
<thead>
<tr>
<th>Command</th>
<th>Action</th>
<th>Sending</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAS_SetParameter</td>
<td>Specific parameter values are saved to the RAM.</td>
<td>5 bytes</td>
<td>1 byte</td>
</tr>
<tr>
<td>FAS_GetParameter</td>
<td>Specific parameter values in the RAM are recognized</td>
<td>1 byte</td>
<td>5 bytes</td>
</tr>
<tr>
<td>FAS_SetIOOutput</td>
<td>Set output signal level of the control output port.</td>
<td>8 bytes</td>
<td>1 byte</td>
</tr>
<tr>
<td>FAS_SetIOInput</td>
<td>Set input signal level of the control input port.</td>
<td>8 bytes</td>
<td>1 byte</td>
</tr>
</tbody>
</table>

### FAS_SetParameter
- **Sending**: 5 bytes
  - 1 byte Parameter number (0~28)
  - 4 bytes Parameter value
- **Response**: 1 byte
  - 1 byte Communication status

Refer to 「1-2-2.Parameter List」

### FAS_GetParameter
- **Sending**: 1 byte
  - 1 byte Parameter number (0~28)
- **Response**: 5 bytes
  - 1 byte Communication status
  - 4 bytes Parameter value

Refer to 「1-2-2.Parameter List」

### FAS_SetIOOutput
- **Sending**: 8 bytes
  - 4 bytes I/O set mask value
  - 4 bytes I/O clear mask value
- **Response**: 1 byte
  - 1 byte Communication status

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」.

### FAS_SetIOInput
- **Sending**: 8 bytes
  - 4 bytes I/O set mask value
  - 4 bytes I/O clear mask value
- **Response**: 1 byte
  - 1 byte Communication status

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」.
### FAS_GetIOInput

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x22</td>
<td>Current input signal status of the control input port is recognized.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sending: 0 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response: 5 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte 4 bytes</td>
<td>Communication status Input status value</td>
</tr>
</tbody>
</table>

Relevant bit by each input signal, refer to 「1-2-4.Bit setup of Input Pin」.

### FAS_GetIOOutput

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x23</td>
<td>Current output signal status of the control output port is recognized.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sending: 0 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response: 5 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte 4 bytes</td>
<td>Communication status Output status value</td>
</tr>
</tbody>
</table>

Relevant bit by each output signal, refer to 「1-2-3.Bit setup of Output Pin」.

### FAS_SetIOAssignMap

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x24</td>
<td>Assign I/O signal to the pin of CN1 port and set signal level simultaneously.</td>
<td>By running 'FAS_SaveAllParameters', you can save the setting value to the ROM.</td>
</tr>
<tr>
<td></td>
<td>Sending: 6 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte 4 bytes 1 byte</td>
<td>I/O number I/O pin masking data Setting level</td>
</tr>
</tbody>
</table>

- I/O number: ‘0-11’ corresponds to ‘Limit+, Limit-, Org, IN1,…, IN9’ respectively, and ‘12-22’ corresponds to ‘COMP, OUT1,…, OUT9’ respectively.
- I/O pin masking data: Refer to 「1-2-4.Bit setup of Input Pin」.
- Level Setting: 0:Active Low, 1:Active High

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x25</td>
<td>Recognize pin setting status of CN1 port from RAM.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sending: 1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>I/O number</td>
</tr>
</tbody>
</table>

- I/O number: ‘0-11’ corresponds to ‘Limit+, Limit-, Org, IN1,…, IN9’ respectively, and ‘12-22’ corresponds to ‘COMP, OUT1,…, OUT9’ respectively.

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x25</td>
<td>Recognize pin setting status of CN1 port from RAM.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sending: 1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>I/O number</td>
</tr>
</tbody>
</table>

- I/O number: ‘0-11’ corresponds to ‘Limit+, Limit-, Org, IN1,…, IN9’ respectively, and ‘12-22’ corresponds to ‘COMP, OUT1,…, OUT9’ respectively.

<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x25</td>
<td>Recognize pin setting status of CN1 port from RAM.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sending: 1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>I/O number</td>
</tr>
</tbody>
</table>

- I/O number: ‘0-11’ corresponds to ‘Limit+, Limit-, Org, IN1,…, IN9’ respectively, and ‘12-22’ corresponds to ‘COMP, OUT1,…, OUT9’ respectively.

Response: 6 bytes

| 1 byte 4 byte 1 byte | Communication status 10 pin masking status Level status |

For more information, refer to 「0x24' Frame type.'
<table>
<thead>
<tr>
<th>Code</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x26</td>
<td>FAS_I0AssignMapReadROM</td>
<td>Recognize setting status of I/O and setting value of signal level from ROM area. These values are loaded to RAM.</td>
</tr>
<tr>
<td></td>
<td>Sending: 0 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response: 2 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>Command performing status</td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>(0: complete, values except 0: error)</td>
</tr>
<tr>
<td></td>
<td>Communication status</td>
<td></td>
</tr>
<tr>
<td>0x27</td>
<td>FAS_TriggerOutput_RunA</td>
<td>Start/Stop command for ‘Compare Out’ signal</td>
</tr>
<tr>
<td></td>
<td>Sending: 18 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>Output start/stop</td>
</tr>
<tr>
<td></td>
<td>4 bytes</td>
<td>Pulse start position</td>
</tr>
<tr>
<td></td>
<td>4 byte</td>
<td>Pulse period</td>
</tr>
<tr>
<td></td>
<td>4 byte</td>
<td>Pulse width</td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>Output pin number</td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>(fix to 0)</td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>spare</td>
</tr>
<tr>
<td></td>
<td>Response: 2 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>Command performing status</td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>(0: complete, values except 0: error)</td>
</tr>
<tr>
<td></td>
<td>Communication status</td>
<td></td>
</tr>
<tr>
<td>0x28</td>
<td>FAS_TriggerOutput_Status</td>
<td>Command to check if the trigger output pulse is working or not.</td>
</tr>
<tr>
<td></td>
<td>Sending: 0 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response: 2 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>Status (1:output ON, 0:output OFF)</td>
</tr>
<tr>
<td>0x2A</td>
<td>FAS_StepEnable</td>
<td>Step ON/OFF status is set.</td>
</tr>
<tr>
<td></td>
<td>Sending: 1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>0:OFF, 1:ON</td>
</tr>
<tr>
<td></td>
<td>Response: 1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>Communication status</td>
</tr>
<tr>
<td></td>
<td>FAS_StepAlarmReset</td>
<td>Reset STEP alarm status or release reset. To make a reset, send 'reset ON' and 'reset release' sequentially.</td>
</tr>
<tr>
<td>---</td>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Sending: 1 byte</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reset ON(1) Reset release(0)</td>
</tr>
<tr>
<td></td>
<td>Response: 1 byte</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>FAS_ServoAlarmtype</th>
<th>To request the Alarm type</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sending: 0 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response: 2 byte</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Alarm type (1~)</td>
</tr>
</tbody>
</table>

- Alarm type: No alarm (0) OverCurrent(1) OverSpeed(2) StepOut(3) OverTemperature(5) BackEMF(6) MotorConnect(7) MotorPower(9) Inposition(10)
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x31 (49)</td>
<td>FAS_MoveStop</td>
<td>Request to stop motor currently operates. Sending: 0 byte. Response: 1 byte.</td>
</tr>
<tr>
<td>0x32 (50)</td>
<td>FAS_EmergencyStop</td>
<td>Request emergency stop of the motor. Sending: 0 byte. Response: 1 byte.</td>
</tr>
<tr>
<td>0x33 (51)</td>
<td>FAS_MoveOriginSingleAxis</td>
<td>Request the motor to return to origin under current setting parameter condition. Sending: 0 byte. Response: 1 byte.</td>
</tr>
<tr>
<td>0x34 (52)</td>
<td>FAS_MoveSingleAxisAbsPos</td>
<td>Request the motor to move its position as much as the absolute value[pulse]. Sending: 8 bytes. Response: 1 byte.</td>
</tr>
<tr>
<td>0x35 (53)</td>
<td>FAS_MoveSingleAxisIncPos</td>
<td>Request the motor to move its position as much as the incremental value[pulse]. Sending: 8 bytes. Response: 1 byte.</td>
</tr>
<tr>
<td>Code</td>
<td>Request Details</td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>----------------</td>
<td></td>
</tr>
<tr>
<td>0x36 (54)</td>
<td>FAS_MoveToLimit Request the motor to start limit motion under current setting parameter condition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sending: 5 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 bytes</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td>Running speed [pps]</td>
<td>Running direction (0: -Limit 1: +Limit)</td>
</tr>
<tr>
<td></td>
<td>Response: 1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Communication status</td>
<td></td>
</tr>
<tr>
<td>0x37 (55)</td>
<td>FAS_MoveVelocity Request the motor to start jog motion at the current setting parameter condition</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sending: 5 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 bytes</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td>Running speed [pps]</td>
<td>Running direction (0: -Jog 1: +Jog)</td>
</tr>
<tr>
<td></td>
<td>Response: 1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Communication status</td>
<td></td>
</tr>
<tr>
<td>0x38 (56)</td>
<td>FAS_PositionAbsOverride Request the motor to change the target absolute position value[pulse] while it is in running.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sending: 4 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Changed command position value [pulse]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response: 1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Communication status</td>
<td></td>
</tr>
<tr>
<td>0x39 (57)</td>
<td>FAS_PositionIncOverride Request the motor to change the target incremental position value[pulse] during operation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sending: 4 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Changed command position value [pulse]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response: 1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Communication status</td>
<td></td>
</tr>
<tr>
<td>0x3A (58)</td>
<td>FAS_VelocityOverride Request the motor to change the running speed value[pps] during operation.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Sending: 4 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Changed running speed [pps]</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Response: 1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Communication status</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Description</td>
<td>Details</td>
</tr>
<tr>
<td>--------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0x3B</td>
<td>FAS_AllMoveStop Request stop for all motor that connected in same port during operation.</td>
<td></td>
</tr>
<tr>
<td>(59)</td>
<td>Sending: 0 byte</td>
<td>(Slave number must be ‘99’)</td>
</tr>
<tr>
<td></td>
<td>Response: no response</td>
<td>(All slaves do not send response because cannot receive response from all slaves simultaneously.)</td>
</tr>
<tr>
<td>0x3C</td>
<td>FAS_AllEmergencyStop Request emergency stop for all motor that connected in same port during operation.</td>
<td></td>
</tr>
<tr>
<td>(60)</td>
<td>Sending: 0 byte</td>
<td>(Slave number must be ‘99’)</td>
</tr>
<tr>
<td></td>
<td>Response: no response</td>
<td>(All slaves do not send response because cannot receive response from all slaves simultaneously.)</td>
</tr>
<tr>
<td>0x3D</td>
<td>FAS_AllMoveOriginSingleAxis Request return to origin under current setting parameter condition for all drives that connected in same port.</td>
<td></td>
</tr>
<tr>
<td>(61)</td>
<td>Sending: 0 byte</td>
<td>(Slave number must be ‘99’)</td>
</tr>
<tr>
<td></td>
<td>Response: no response</td>
<td>(All slaves do not send response because cannot receive response from all slaves simultaneously.)</td>
</tr>
<tr>
<td>0x3E</td>
<td>FAS_AllSingleAxisAbsPos Request move its position as much as the absolute value[pulse] for all drives that connected in same port.</td>
<td></td>
</tr>
<tr>
<td>(62)</td>
<td>Sending: 8 bytes (Slave number must be ‘99’)</td>
<td>4 bytes Absolute position value 4 bytes Running speed [pps]</td>
</tr>
<tr>
<td></td>
<td>Response: no response</td>
<td>(All slaves do not send response because cannot receive response from all slaves simultaneously.)</td>
</tr>
<tr>
<td>0x3F</td>
<td>FAS_AllSingleAxisIncPos Request move its position as much as the incremental value[pulse] for all drives that connected in same port.</td>
<td></td>
</tr>
<tr>
<td>(63)</td>
<td>Sending: 8 bytes (Slave number must be ‘99’)</td>
<td>4 bytes incremental position value 4 bytes Running speed [pps]</td>
</tr>
<tr>
<td></td>
<td>Response: no response</td>
<td>(All slaves do not send response because cannot receive response from all slaves simultaneously.)</td>
</tr>
<tr>
<td>0x80</td>
<td>FAS_MoveSingleAxisAbsPos Ex Request the motor to move its position as much as the absolute value[pulse] with Custom Accel. / Decel. Time[msec]</td>
<td></td>
</tr>
<tr>
<td>(128)</td>
<td>Sending: 40 bytes</td>
<td>Absolute position value 4 bytes Running speed [pps] Flag option 2 bytes Custom Accel. Time (1-9999)</td>
</tr>
</tbody>
</table>
### Communication Protocols

**Flag option**: 0x0001: reserved  
0x0002: Custom Accel. Time is used.  
0x0004: Custom Decel. Time is used.  
If the Flag bit is OFF status(0), Accel./Decel. time value is used that saved in controller.

**Response**: 1 byte

<table>
<thead>
<tr>
<th>2 bytes</th>
<th>24 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom Decel. Time</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Request the motor to move its position as much as the absolute value[pulse] with Custom Accel. / Decel. Time[msec]

**Sending**: 40 bytes

- 4 bytes: Incremental position value
- 4 bytes: Running speed [pps]
- 4 bytes: Flag option
- 2 bytes: Custom Accel. Time (1-9999)

**Flag option**: 0x0001: reserved  
0x0002: Custom Accel. Time is used.  
0x0004: Custom Decel. Time is used.  
If the Flag bit is OFF status(0), Accel./Decel. time value is used that saved in controller.

**Response**: 1 byte

<table>
<thead>
<tr>
<th>2 bytes</th>
<th>24 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom Decel. Time</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

Request the motor to start jog motion at the current setting parameter condition with custom Accel/Decel time value[msec].

**Sending**: 37 bytes

- 4 bytes: Running speed [pps]
- 1 byte: Running direction (0: -Jog, 1: +Jog)
- 4 bytes: Flag option

<table>
<thead>
<tr>
<th>2 bytes</th>
<th>26 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Custom Accel./Decel. Time</td>
<td>Reserved</td>
</tr>
</tbody>
</table>

**Flag option**: 0x0001: reserved  
0x0002: Custom Accel./Decel. Time is used.  
If the Flag bit is OFF status(0), Accel./Decel. time value is used that saved in controller.

**Response**: 1 byte
| FAS_MoveLinearAbsPos | Fulfill Linear Interpolation for multi-drives connected in same port. Position value is absolute value [pulse] unit and refer to «2. Library for PC Program».

FAS_MoveLinearIncPos | Fulfill Linear Interpolation for multi-drives connected in same port. Position value is incremental value [pulse] unit and refer to «2. Library for PC Program».

| FAS_GetAxisStatus | Request the flag value indicates operation status

| Sending : 0 byte
| Response : 5 bytes

| 1 byte | 4 bytes |
| Communication status | Status flag value |

Assign bit related to each Flag, refer to «1-2-5. Bit setup of Status Flag».

| FAS_GetIOAxisStatus | Request the I/O status and the running Flag status.

| Sending : 0 byte
| Response : 13 bytes

| 1 byte | 4 bytes | 4 bytes | 4 bytes |
| Communication status | Input status value | Output status value | Status flag value |

| FAS_GetMotionStatus | Request the current operation progress status and its Position Table number

| Sending : 0 byte
| Response : 21 bytes

| 1 byte | 4 bytes | 4 bytes | 4 bytes | 4 bytes | 4 bytes |
| Communication status | Command position value | Actual Position value | Position Difference value | Running speed value | Current running PT number |

*Actual Position value : when external encoder is connected
FAS_GetAllStatus

Request all data including the current running status
(Frame type 0x41 and 0x42 are packed.)

Sending: 0 byte
Response: 33 bytes

<table>
<thead>
<tr>
<th>1 byte</th>
<th>4 bytes</th>
<th>4 bytes</th>
<th>4 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication status</td>
<td>Input status value</td>
<td>Output status value</td>
<td>Status flag value</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>4 bytes</th>
<th>4 bytes</th>
<th>4 bytes</th>
<th>4 bytes</th>
<th>4 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command position value</td>
<td>Actual position value</td>
<td>Position Difference value</td>
<td>Running speed value</td>
<td>Current running PT number</td>
</tr>
</tbody>
</table>

*Actual Position value: when external encoder is connected

FAS_SetCommandPos

User can set the command position value before it starts and then can check how the command position value is changed.

Sending: 4 bytes

<table>
<thead>
<tr>
<th>4 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command position setting count value</td>
</tr>
</tbody>
</table>

Response: 1 byte

<table>
<thead>
<tr>
<th>1 byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication status</td>
</tr>
</tbody>
</table>

FAS_GetCommandPos

Request the command position value[pulse] being tracked.

Sending: 0 byte
Response: 5 bytes

<table>
<thead>
<tr>
<th>1 byte</th>
<th>4 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication status</td>
<td>Command position value</td>
</tr>
</tbody>
</table>

FAS_SetActualPos

When external encoder is connected to drive, the actual position value is continuously renewed while the motor is operating. User can set the actual position value before it starts and then can check how the actual position value is changed.

Sending: 4 bytes

<table>
<thead>
<tr>
<th>4 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Actual position count value</td>
</tr>
</tbody>
</table>

Response: 1 byte

<table>
<thead>
<tr>
<th>1 byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication status</td>
</tr>
</tbody>
</table>

FAS_GetActualPos

Request the current actual position value[pulse].
* When external encoder is connected

Sending: 0 byte
Response: 5 bytes

<table>
<thead>
<tr>
<th>1 byte</th>
<th>4 bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication status</td>
<td>Actual position value</td>
</tr>
</tbody>
</table>
### Communication Protocols

<table>
<thead>
<tr>
<th>Code</th>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x54</td>
<td>FAS_GetPosError</td>
<td>Request the difference[pulse] between the command position value and the actual position value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>*When external encoder is connected.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 0 byte, Response: 5 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>0x55</td>
<td>FAS_GetActualVel</td>
<td>Request the current running speed value [pps]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 0 byte, Response: 5 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>0x56</td>
<td>FAS_ClearPosition</td>
<td>User can set the command position and actual position value as '0' before it starts to operate and can check how the command position value is changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 0 byte, Response: 1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>0x58</td>
<td>FAS_MovePause</td>
<td>To request the pause start and pause end of motor motioning.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td>0x60</td>
<td>FAS_PosTableReadItem</td>
<td>To read Position Table values in the RAM of the drive.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 2 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 65 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td><img src="image" alt="Diagram" /></td>
</tr>
</tbody>
</table>

For items by each PT, refer to 「1-2-6. Position Table Item」.
<table>
<thead>
<tr>
<th>Code</th>
<th>Command Name</th>
<th>Description</th>
<th>Sending</th>
<th>Response</th>
<th>Communication status</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x61</td>
<td>FAS_PosTableWriteItem</td>
<td>To save Position Table values to the RAM of the drive.</td>
<td>66 bytes</td>
<td>2 bytes</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 66 bytes</td>
<td></td>
<td></td>
<td>Command performing status (values except 0: complete, 0: error)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For items by each PT, refer to 「1-2-6. Position Table Item」.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 2 bytes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x62</td>
<td>FAS_PosTableReadROM</td>
<td>To read all Position Table values (256 ea) in the ROM of the drive.</td>
<td>0 byte</td>
<td>2 bytes</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 0 byte</td>
<td></td>
<td></td>
<td>Command performing status (0: complete, values except 0: error)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 2 bytes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x63</td>
<td>FAS_PosTableWriteROM</td>
<td>To save all Position Table value(256 ea) to the ROM of the drive.</td>
<td>0 byte</td>
<td>2 bytes</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 0 byte</td>
<td></td>
<td></td>
<td>Command performing status (0: complete, values except 0: error)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 2 bytes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x64</td>
<td>FAS_PosTableRunItem</td>
<td>To start the position table operation from the designated Position Table number</td>
<td>2 bytes</td>
<td>1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 2 bytes</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x6A</td>
<td>FAS_PosTableReadOneItem</td>
<td>To read one of Position Table values in the RAM of the drive.</td>
<td>4 byte</td>
<td>4 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 4 byte</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Refer to 「1-2-6. Position Table Item」 for Offset value
**Frame Type '0x65' ~ '0x69', '0x0E' ~ '0x0F' are assigned for internal use.**

* PT Number : 0-255 for Ezi-STEP-PR
  0-63 for Ezi-STEP-PR-MI/Ezi-STEP-ALL
1–2–2. Parameter Lists

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Unit</th>
<th>Lower Limit</th>
<th>Upper Limit</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Pulse per Revolution</td>
<td></td>
<td>0</td>
<td>15</td>
<td>10</td>
</tr>
<tr>
<td>1</td>
<td>Axis Max Speed</td>
<td>[pps]</td>
<td>1</td>
<td>500,000</td>
<td>500,000</td>
</tr>
<tr>
<td>2</td>
<td>Axis Start Speed</td>
<td>[pps]</td>
<td>1</td>
<td>35,000</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>Axis Acc Time</td>
<td>[msec]</td>
<td>1</td>
<td>9.999</td>
<td>100</td>
</tr>
<tr>
<td>4</td>
<td>Axis Dec Time</td>
<td>[msec]</td>
<td>1</td>
<td>9999</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>Speed Override</td>
<td>[%]</td>
<td>1</td>
<td>500</td>
<td>100</td>
</tr>
<tr>
<td>6</td>
<td>Jog Speed</td>
<td>[pps]</td>
<td>1</td>
<td>500,000</td>
<td>5,000</td>
</tr>
<tr>
<td>7</td>
<td>Jog Start Speed</td>
<td>[pps]</td>
<td>1</td>
<td>35,000</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>Jog Acc Dec Time</td>
<td>[msec]</td>
<td>1</td>
<td>9.999</td>
<td>100</td>
</tr>
<tr>
<td>9</td>
<td>Servo Alarm Logic</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>Servo ON Logic</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>11</td>
<td>Servo Alarm Reset Logic</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>S/W Limit Stop Method</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>15</td>
<td>H/W Limit Stop Method</td>
<td></td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>16</td>
<td>Limit Sensor Logic</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>17</td>
<td>Org Speed</td>
<td>[pps]</td>
<td>1</td>
<td>500,000</td>
<td>5,000</td>
</tr>
<tr>
<td>18</td>
<td>Org Search Speed</td>
<td>[pps]</td>
<td>1</td>
<td>500,000</td>
<td>1,000</td>
</tr>
<tr>
<td>19</td>
<td>Org Acc Dec Time</td>
<td>[msec]</td>
<td>1</td>
<td>9.999</td>
<td>50</td>
</tr>
<tr>
<td>20</td>
<td>Org Method</td>
<td></td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>21</td>
<td>Org Dir</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>22</td>
<td>Org Offset</td>
<td>[pulse]</td>
<td>-134,217,727</td>
<td>+134,217,727</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>Org Position Set</td>
<td>[pulse]</td>
<td>-134,217,727</td>
<td>+134,217,727</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>Org Sensor Logic</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>25</td>
<td>Stop current</td>
<td>[%]</td>
<td>20</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>26</td>
<td>Motion Dir</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>Limit Sensor Dir</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>28</td>
<td>Encoder Multiply Value</td>
<td></td>
<td>0</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>EncoderDir</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>Pos. Value Counting Method</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
1-2-3. Setup bit of Output pin

Here is detail description of ‘0x20’ frame type. This command is only applicable to 9 signals of ‘User Output 0’ ~ ‘User Output 8’ out of 24 signal types in the control output port. The rest of 15 output signals cannot be operated by the user’s disposal. When any relevant situation occurs while the drive operates, they will be indicated. The following table shows bit mask values by each signal.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Relevant Bit Position</th>
<th>Signal Name</th>
<th>Relevant Bit Position</th>
<th>Signal Name</th>
<th>Relevant Bit Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare Out</td>
<td>0x00000001</td>
<td>Origin Search OK</td>
<td>0x00000100</td>
<td>User Output 1</td>
<td>0x00010000</td>
</tr>
<tr>
<td>reserved</td>
<td>0x00000002</td>
<td>reserved</td>
<td>0x00000200</td>
<td>User Output 2</td>
<td>0x00020000</td>
</tr>
<tr>
<td>Alarm</td>
<td>0x00000004</td>
<td>reserved</td>
<td>0x00000400</td>
<td>User Output 3</td>
<td>0x00040000</td>
</tr>
<tr>
<td>Moving</td>
<td>0x00000008</td>
<td>reserved</td>
<td>0x00000800</td>
<td>User Output 4</td>
<td>0x00080000</td>
</tr>
<tr>
<td>Acc/Dec</td>
<td>0x00000010</td>
<td>PT Output 0</td>
<td>0x00001000</td>
<td>User Output 5</td>
<td>0x00100000</td>
</tr>
<tr>
<td>ACK</td>
<td>0x00000020</td>
<td>PT Output 1</td>
<td>0x00002000</td>
<td>User Output 6</td>
<td>0x00200000</td>
</tr>
<tr>
<td>END</td>
<td>0x00000040</td>
<td>PT Output 2</td>
<td>0x00004000</td>
<td>User Output 7</td>
<td>0x00400000</td>
</tr>
<tr>
<td>AlarmBlink</td>
<td>0x00000080</td>
<td>User Output 0</td>
<td>0x00008000</td>
<td>User Output 8</td>
<td>0x00800000</td>
</tr>
</tbody>
</table>

**Example 1** Sending data to turn ON the User Output 5.

<table>
<thead>
<tr>
<th>4 bytes (I/O set mask value)</th>
<th>4 bytes (I/O clear mask value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00100000</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

**Example 2** Sending data to turn OFF the User Output 5.

<table>
<thead>
<tr>
<th>4 bytes (I/O set mask value)</th>
<th>4 bytes (I/O clear mask value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>0x00100000</td>
</tr>
</tbody>
</table>

1-2-4. Setup bit of Input pin

Here is detail description of ‘0x21’ frame type. This command is only applicable to 32 signals in the control input port. User can use signals for testing as if they are inputted without actual input signal. The following table shows bit mask values by each signal.

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Relevant Bit Position</th>
<th>Signal Name</th>
<th>Relevant Bit Position</th>
<th>Signal Name</th>
<th>Relevant Bit Position</th>
<th>Signal Name</th>
<th>Relevant Bit Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limit+</td>
<td>0x00000001</td>
<td>PT A4</td>
<td>0x00000100</td>
<td>AlarmReset</td>
<td>0x00001000</td>
<td>JPT input 2</td>
<td>0x01000000</td>
</tr>
<tr>
<td>Limit-</td>
<td>0x00000002</td>
<td>PT A5</td>
<td>0x00000200</td>
<td>reserved</td>
<td>0x00020000</td>
<td>JPT Start</td>
<td>0x02000000</td>
</tr>
<tr>
<td>Origin</td>
<td>0x00000004</td>
<td>PT A6</td>
<td>0x00000400</td>
<td>Pause</td>
<td>0x00040000</td>
<td>User Input 0</td>
<td>0x04000000</td>
</tr>
<tr>
<td>Clear Position</td>
<td>0x00000008</td>
<td>PT A7</td>
<td>0x00000800</td>
<td>Org Search</td>
<td>0x00080000</td>
<td>User Input 1</td>
<td>0x08000000</td>
</tr>
<tr>
<td>PT A0</td>
<td>0x00000010</td>
<td>PT Start</td>
<td>0x00001000</td>
<td>Teaching</td>
<td>0x00100000</td>
<td>User Input 2</td>
<td>0x10000000</td>
</tr>
<tr>
<td>PT A1</td>
<td>0x00000020</td>
<td>Stop</td>
<td>0x00002000</td>
<td>E-stop</td>
<td>0x00200000</td>
<td>User Input 3</td>
<td>0x20000000</td>
</tr>
<tr>
<td>PT A2</td>
<td>0x00000040</td>
<td>Jog+</td>
<td>0x00004000</td>
<td>JPT input 0</td>
<td>0x00400000</td>
<td>User Input 4</td>
<td>0x40000000</td>
</tr>
<tr>
<td>PT A3</td>
<td>0x00000080</td>
<td>Jog-</td>
<td>0x00008000</td>
<td>JPT input 1</td>
<td>0x00800000</td>
<td>User Input 5</td>
<td>0x80000000</td>
</tr>
</tbody>
</table>
Example 1] Sending data to turn ON the Pause port

<table>
<thead>
<tr>
<th>4 bytes (I/O set mask value)</th>
<th>4 bytes (I/O clear mask value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00040000</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

Example 2] Sending data to turn OFF the Pause port

<table>
<thead>
<tr>
<th>4 bytes (I/O set mask value)</th>
<th>4 bytes (I/O clear mask value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x00000000</td>
<td>0x00040000</td>
</tr>
</tbody>
</table>

1–2–5. Bit setup of Status Flag
Refer to ‘EZISTEP_AXISSTATUS’ structure of ‘motion_define.h’ of include folder.

<table>
<thead>
<tr>
<th>Name of Flag Define</th>
<th>Contents</th>
<th>Relevant Bit Position</th>
</tr>
</thead>
<tbody>
<tr>
<td>FFLAG_ERRORALL</td>
<td>One or more errors occur.</td>
<td>0x00000001</td>
</tr>
<tr>
<td>FFLAG_HWPOSILMT</td>
<td>‘+’ direction limit sensor turns ON.</td>
<td>0x00000002</td>
</tr>
<tr>
<td>FFLAG_HWNEGALMT</td>
<td>‘-’ direction limit sensor turns ON.</td>
<td>0x00000004</td>
</tr>
<tr>
<td>FFLAG_SWPOSILMT</td>
<td>‘+’ direction program limit is exceeded.</td>
<td>0x00000008</td>
</tr>
<tr>
<td>FFLAG_SWNEGALMT</td>
<td>‘-’ direction program limit is exceeded.</td>
<td>0x00000010</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td>0x00000020</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td>0x00000040</td>
</tr>
<tr>
<td>FFLAG_ERRSTEPALARM</td>
<td>One or more errors of STEP alarm (8 ea) occur.</td>
<td>0x00000080</td>
</tr>
<tr>
<td>FFLAG_ERROVERCURRENT</td>
<td>The motor driving device is under over-current.</td>
<td>0x00000100</td>
</tr>
<tr>
<td>FFLAG_ERROVERSPEED</td>
<td>The motor speed exceeded 3000[rpm].</td>
<td>0x00000200</td>
</tr>
<tr>
<td>FFLAG_ERRSPEED</td>
<td>The motor is not tracked normally by pulse input.</td>
<td>0x00000400</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td>0x00000800</td>
</tr>
<tr>
<td>FFLAG_ERROVERHEAT</td>
<td>The internal temperature of the drive exceeds 55°C.</td>
<td>0x00001000</td>
</tr>
<tr>
<td>FFLAG_ERRPREVPWR</td>
<td>A counter electromotive force of the motor exceeds 70V.</td>
<td>0x00002000</td>
</tr>
<tr>
<td>FFLAG_ERRMOTORPOWER</td>
<td>The motor voltage is abnormal.</td>
<td>0x00004000</td>
</tr>
<tr>
<td>FFLAG_ERPLLOWPOWER</td>
<td>The drive voltage is abnormal.</td>
<td>0x00008000</td>
</tr>
<tr>
<td>FFLAG_EMGSTOP</td>
<td>The motor is under emergency stop.</td>
<td>0x00010000</td>
</tr>
<tr>
<td>FFLAG_SLOWSTOP</td>
<td>The motor is under general stop.</td>
<td>0x00020000</td>
</tr>
<tr>
<td>FFLAG_ORIGINRETURNING</td>
<td>The motor is returning to the origin.</td>
<td>0x00040000</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td>0x00080000</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td>0x00100000</td>
</tr>
<tr>
<td>FFLAG_ALARMRESET</td>
<td>Alarm Reset has run.</td>
<td>0x00200000</td>
</tr>
<tr>
<td>FFLAG_PTSTOPED</td>
<td>Position Table operation has been finished.</td>
<td>0x00400000</td>
</tr>
<tr>
<td>FFLAG_ORIGINSENSOR</td>
<td>The origin sensor is ON.</td>
<td>0x00800000</td>
</tr>
<tr>
<td>FFLAG_ZPULSE</td>
<td>The z-pulse signal of motor is ON status.</td>
<td>0x01000000</td>
</tr>
<tr>
<td>FFLAG_ORIGINAETOK</td>
<td>Origin return operation has been finished.</td>
<td>0x02000000</td>
</tr>
<tr>
<td>FFLAG_MOTIONDIR</td>
<td>To display the motor operating direction (+: OFF, -: ON)</td>
<td>0x04000000</td>
</tr>
<tr>
<td>FFLAG_MOTIONING</td>
<td>The motor is running.</td>
<td>0x08000000</td>
</tr>
<tr>
<td>FFLAG_MOTIONPAUSE</td>
<td>The motor in running is stopped by Pause command.</td>
<td>0x10000000</td>
</tr>
<tr>
<td>FFLAG_MOTIONACCEL</td>
<td>The motor is operating to the acceleration section.</td>
<td>0x20000000</td>
</tr>
<tr>
<td>FFLAG_MOTIONDECEL</td>
<td>The motor is operating to the deceleration section.</td>
<td>0x40000000</td>
</tr>
<tr>
<td>FFLAG_MOTIONCONST</td>
<td>The motor is not running as Acceleration/Deceleration but as constant speed of operation.</td>
<td>0x80000000</td>
</tr>
</tbody>
</table>

1–2–6. Position Table Item
Refer to ‘motion_define.h’ of include files.
### Communication Protocols

<table>
<thead>
<tr>
<th>Name</th>
<th>Name of Structure Parameter</th>
<th>Number of Bytes</th>
<th>Offset position</th>
<th>Unit</th>
<th>Low Limit</th>
<th>Upper Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>lPosition</td>
<td>4 (signed)</td>
<td>0</td>
<td>[pulse]</td>
<td>-134217728</td>
<td>+134217728</td>
</tr>
<tr>
<td>Low Speed</td>
<td>dwStartSpd</td>
<td>4 (unsigned)</td>
<td>4</td>
<td>[pps]</td>
<td>0</td>
<td>500000</td>
</tr>
<tr>
<td>High Speed</td>
<td>dwMoveSpd</td>
<td>4 (unsigned)</td>
<td>8</td>
<td>[pps]</td>
<td>0</td>
<td>500000</td>
</tr>
<tr>
<td>Accel. Time</td>
<td>wAccelRate</td>
<td>2 (unsigned)</td>
<td>12</td>
<td>[msec]</td>
<td>1</td>
<td>9999</td>
</tr>
<tr>
<td>Decel. Time</td>
<td>wDecelRate</td>
<td>2 (unsigned)</td>
<td>14</td>
<td>[msec]</td>
<td>1</td>
<td>9999</td>
</tr>
<tr>
<td>Command</td>
<td>wCommand</td>
<td>2 (unsigned)</td>
<td>16</td>
<td></td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>Wait time</td>
<td>wWaitTime</td>
<td>2 (unsigned)</td>
<td>18</td>
<td>[msec]</td>
<td>0</td>
<td>600000</td>
</tr>
<tr>
<td>Continuous Action</td>
<td>wContinuous</td>
<td>2 (unsigned)</td>
<td>20</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Jump Table No.</td>
<td>wBranch</td>
<td>2 (unsigned)</td>
<td>22</td>
<td></td>
<td>0</td>
<td>10255</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Jump PT 0</td>
<td>wCond_branch0</td>
<td>2 (unsigned)</td>
<td>24</td>
<td></td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10000</td>
<td>10255</td>
</tr>
<tr>
<td>Jump PT 1</td>
<td>wCond_branch1</td>
<td>2 (unsigned)</td>
<td>26</td>
<td></td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10000</td>
<td>10255</td>
</tr>
<tr>
<td>Jump PT 2</td>
<td>wCond_branch2</td>
<td>2 (unsigned)</td>
<td>28</td>
<td></td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10000</td>
<td>10255</td>
</tr>
<tr>
<td>Loop Count</td>
<td>wLoopCount</td>
<td>2 (unsigned)</td>
<td>30</td>
<td></td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Loop Jump Table No.</td>
<td>wBranchAfterLoop</td>
<td>2 (unsigned)</td>
<td>32</td>
<td></td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10000</td>
<td>10255</td>
</tr>
<tr>
<td>PT set</td>
<td>wPTSet</td>
<td>2 (unsigned)</td>
<td>34</td>
<td></td>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>Loop Counter Clear</td>
<td>wLoopCountCLR</td>
<td>2 (unsigned)</td>
<td>36</td>
<td></td>
<td>0</td>
<td>255</td>
</tr>
<tr>
<td>Compare Position</td>
<td>lTriggerPos</td>
<td>4 (signed)</td>
<td>38</td>
<td>[pulse]</td>
<td>-134217728</td>
<td>+134217728</td>
</tr>
<tr>
<td>Compare Width</td>
<td>wTriggerOnTime</td>
<td>2 (unsigned)</td>
<td>42</td>
<td>[msec]</td>
<td>1</td>
<td>9999</td>
</tr>
<tr>
<td>Blank</td>
<td></td>
<td>20 (unsigned)</td>
<td>44</td>
<td>0x00</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

For the setting method by each item, refer to other manual 「User Manual_Position Table」.
Please refer to separate manual 「User Manual_Position Table」 for setting method per each time.

### 1-2-7. Information of Motors

First 2 digits of number and 1-2 characters indicate the motor size and length.

【Example】56XL : Motor Flange size is 56mm and Extra large size

Other part indicates the motor manufacturer information as below.

<table>
<thead>
<tr>
<th>Display</th>
<th>Maker</th>
</tr>
</thead>
<tbody>
<tr>
<td>blank</td>
<td>JapanServo</td>
</tr>
<tr>
<td>SD</td>
<td>Sanyo Denki</td>
</tr>
<tr>
<td>POR</td>
<td>Portescap</td>
</tr>
<tr>
<td>NPM</td>
<td>NPM</td>
</tr>
<tr>
<td>FUL</td>
<td>Fulling</td>
</tr>
<tr>
<td>YK</td>
<td>Yunkong</td>
</tr>
<tr>
<td>MIN</td>
<td>Minebia</td>
</tr>
<tr>
<td>Lin</td>
<td>Linear Step</td>
</tr>
</tbody>
</table>
1-3. Program Method

There are 2 methods of programming for Ezi-STEP Plus-R.

The first is generally used with using Visual C++ language under the window system of PC. Library that services together with Ezi-STEP Plus-R have to be used. Please refer to 「2. Library for PC Program」.

The second method is sending command (characters) directly to Ezi-STEP Plus-R. User has to prepare low-level protocol programming like ‘Protocol Test’ program and this method is applied when use higher-level control unit as like PLC.

For more programming method details, please exercise ‘ProtocolTest_PlusR.exe’ is serviced together with GUI.

Please refer to 「3. Protocol for PLC Program」.
2. Library for PC Program

2-1. Library Configuration

To use this library, C++ header file(*.h) and library file(*.lib or *.dll) are required. These files locate in "\FASTECH\EziMOTION PlusR\include\". And the following contents should be included in a source file for development.

```
#include "\FASTECH\EziMOTION PlusR\include\FAS_EziMotionPlusR.h"
#include "\FASTECH\EziMOTION PlusR\include\COMM_Define.h"
#include "\FASTECH\EziMOTION PlusR\include\MOTION_DEFINE.h"
#include "\FASTECH\EziMOTION PlusR\include\ReturnCodes_Define.h"
```

Also, library files are as follows:

```
"\FASTECH\EziMOTION PlusR\include\EziMotionPlusR.lib"
"\FASTECH\EziMOTION PlusR\include\EziMotionPlusR.dll"
```

A sample program source of with using these libraries locate at "\FASTECH\EziMOTION PlusR\Examples\" folder.

(1) The following table explains values returned when each library(DLL) function is used. The user can only check the values returned at the library(DLL) function. Low level programming method does not support following table.

<table>
<thead>
<tr>
<th>Item</th>
<th>Definition</th>
<th>Returned Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>FMM_OK</td>
<td>0</td>
<td>The function has normally performed the command.</td>
</tr>
<tr>
<td>Input Error</td>
<td>FMM_NOT_OPEN</td>
<td>1</td>
<td>Wrong port number is inputted.</td>
</tr>
<tr>
<td></td>
<td>FMM_INVALID_PORT_NUM</td>
<td>2</td>
<td>The port that is not connected.</td>
</tr>
<tr>
<td></td>
<td>FMM_INVALID_SLAVE_NUM</td>
<td>3</td>
<td>Wrong slave number is inputted.</td>
</tr>
<tr>
<td>Operation Error</td>
<td>FMM_POSTABLE_ERROR</td>
<td>9</td>
<td>An error occurs while the motor accesses to the position table.</td>
</tr>
<tr>
<td>Connection Error</td>
<td>FMC_DISCONNECTED</td>
<td>5</td>
<td>The relevant drive is disconnected.</td>
</tr>
<tr>
<td></td>
<td>FMC_TIMEOUT_ERROR</td>
<td>6</td>
<td>Response delay(100 msec) occurs.</td>
</tr>
<tr>
<td></td>
<td>FMC_CRCFAILED_ERROR</td>
<td>7</td>
<td>Checksum error occurs.</td>
</tr>
<tr>
<td></td>
<td>FMC_RECV_PACKET_ERROR</td>
<td>8</td>
<td>Protocol Level error occurs in packet that comes from Drive.</td>
</tr>
</tbody>
</table>
(2) The following table indicates return values included commonly in all libraries and these functions offer to check the result (communication status, running status) judged by the drive. These functions are available for using library (DLL) and protocol.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
<th>Returned Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>FMP_OK</td>
<td>0</td>
<td>Communication has been normally performed.</td>
</tr>
<tr>
<td>Input Error</td>
<td>FMP_FRAMETYPEERROR</td>
<td>128</td>
<td>The drive cannot recognize the command.</td>
</tr>
<tr>
<td></td>
<td>FMP_DATAERROR</td>
<td>129</td>
<td>Input data is out of the range.</td>
</tr>
<tr>
<td>Operation Error</td>
<td>FMP_BUSY MOTOR</td>
<td>133</td>
<td>The motor is already running or not prepared for running.</td>
</tr>
<tr>
<td>Connection Error</td>
<td>FMP_PACKETERROR</td>
<td>130</td>
<td>Protocol level error occurs in packet that Drive’s received.</td>
</tr>
<tr>
<td></td>
<td>FMP_PACKET_CRC_ERROR</td>
<td>170</td>
<td>CRC value is not correct in packet that Drive’s received.</td>
</tr>
</tbody>
</table>

2-2. Communication Status Window

Above communication status is divided by 3 groups.

(1) Communication Error

FMM_NOT_OPEN,

COM Port is not connected. (This error cannot be occurred in GUI.)

FMM_INVALID_PORT_NUM,

COM Port number does not exist. Checking the ‘Device Manager’ window in Window OS. (This error cannot be occurred in GUI.)
Slave number does not exist. Checking the ID value of the drive. (This error cannot be occurred in GUI.)

FMC_DISCONNECTED = 5,

COM Port is disconnected during communication. Checking the communication cable or Power of the drive.

FMC_TIMEOUT_ERROR,

There is no response from the drive.

FMC_CRCFAILED_ERROR,

CRC value of communication packet from the drive is not correct. Checking the possibility of noise on communication cable.

FMC_RECV PACKET_ERROR,
The length of received packet is not correct. Checking the possibility of noise on communication cable.

FMP_FRAMETYPEERROR = 0x80,

Drive does not recognize the command or wrong command is sent. Checking the command value that you want to send to the drive.

FMP_DATAERROR,

The value of the sent data is out of the proper range of the drive. Checking the value that you want to send to the drive.

FMP_PACKETERROR,

The length of received packet on drive is not correct. Checking the possibility of noise on communication cable.

FMP_PACKETCRCERROR = 0xAA,

The incorrect CRC value of packet sent to the drive. Checking the possibility of noise on communication cable.
(2) Wrong Command

FMP_RUNFAIL = 0x85,
Fail on motion command: Tried to new motion under following status.
- The motor is already running
- The motor is under stop command
- Try to Z-pulse Origin without external encoder (only for Ezi-STEP)

FMP_RESETFAIL,
Fail on reset command: Tried to new motion under following status.
- Already 'Reset' status by external input signal.

(3) Command Execution Error

FMM_POSTABLE_ERROR,
The execution of DLL library for 'Position Table' is failed.
### 2-3. Drive Link Function

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAS_Connect</td>
<td>The drive tries to connect communication with the drive module: When it is successfully connected, TRUE will be returned. Otherwise, FALSE will be returned.</td>
</tr>
<tr>
<td>FAS_Close</td>
<td>The drive tries to disconnect communication with the drive module.</td>
</tr>
<tr>
<td>FAS_GetSlaveInfo</td>
<td>The drive reads drive type and program version: Drive type and version information will be returned.</td>
</tr>
<tr>
<td>FAS_GetMotorInfo</td>
<td>The drive reads motor type and manufacturer information: Motor type and maker information will be returned.</td>
</tr>
<tr>
<td>FAS_IsSlaveExist</td>
<td>Check the existence of the relevant drive: When it exists, TRUE will be returned. Otherwise, FALSE will be returned.</td>
</tr>
<tr>
<td>FAS_EnableLog</td>
<td>To select the communication error log function ON/OFF : When it exists, TRUE will return. Otherwise, FALSE will return.</td>
</tr>
<tr>
<td>FAS_SetLogPath</td>
<td>To set the saved folder name of error log file: When folder exists, TRUE will return. Otherwise, FALSE will return.</td>
</tr>
</tbody>
</table>
**FAS_Connect**

FAS_Connect is the function of connection Ezi-STEP Plus-R.

**Syntax**

```c
BOOL FAS_Connect(
    BYTE nPortNo,
    DWORD dwBaud
);
```

**Parameters**

- `nPortNo`
  - Select a serial port number to be connected.
- `dwBaud`
  - Input the Baudrate of the serial port.

**Return Value**

- When it is successfully connected, TRUE will be returned. Otherwise, FALSE will be returned.

**Remarks**

**Example**

```c
#include "FAS_EziMOTIONPlusR.h"

void funcInit()
{
    BYTE nPortNo = 1; // COMM Port Number
    DWORD dwBaudrate = 115200; // Baudrate. (Be variable by setting)
    BYTE iSlaveNo = 0; // Slave No (0 ~ 15)
    char lpBuff[256]:
    int nBuffSize = 256:
    BYTE nType:
    int nRtn:

    // Try to connect
    if (FAS_Connect(nPortNo, dwBaudrate) == FALSE)
    {
        // Connection failed.
        // The port is not connected or the baudrate may be wrong.
        return;
    }

    if (FAS_IsSlaveExist(nPortNo, iSlaveNo) == FALSE)
    {
        // There is no relevant slave number.
        // Check the slave number of Ezi-STEP Plus-R.
        return;
    }

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

    printf("Port : %d (Slave %d) \n", nPortNo, iSlaveNo):
    printf("Type : %d \n", nType):
    printf("Version : %d\n", (lpBuff):
```
// Disconnect.
FAS_Close(nPortNo):
}

See Also
FAS_Close
**FAS_Close**

To disconnect the serial port being used

**Syntax**

```c
void FAS_Close(
    BYTE nPortNo
);
```

**Parameters**

- **nPortNo**
  
  Port number to be disconnected

**Remarks**

**Example**

Refer to ‘FAS_Connect’ library.

**See Also**

FAS_Connect
**FAS_GetSlaveInfo**

To get the version information string of the relevant drive

Syntax

```c
int FAS_GetSlaveInfo(
    BYTE nPortNo,
    BYTE iSlaveNo,
    BYTE* pType,
    LPSTR lpBuff,
    int nBuffSize
);
```

Parameters

- `nPortNo` Port number of relevant drive
- `iSlaveNo` Slave number of relevant drive
- `pType` Type number of relevant drive
- `lpBuff` Buffer pointer will get version information string
- `nBuffSize` Memory allocation size of `lpBuff`

Return Value

- `FMM_OK` : Command has been successfully performed.
- `FMM_NOT_OPEN` : The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM` : There is no `nPort` in the connected ports.
- `FMM_INVALID_SLAVE_NUM` : There is no drive of `iSlaveNo` in the relevant port.

Remarks

Example

Refer to ‘FAS_Connect’ library.

See Also
FAS_GetMotorInfo

To get the motor information string of the relevant drive

Syntax

```c
int FAS_GetMotorInfo(
    BYTE nPortNo,
    BYTE iSlaveNo,
    BYTE* pType,
    LPSTR lpBuff,
    int nBuffSize
);
```

Parameters

- `nPortNo`: Port number of relevant drive
- `iSlaveNo`: Slave number of relevant drive
- `pType`: Type number of relevant motor
- `lpBuff`: Buffer pointer to get version information string
- `nBuffSize`: Memory allocation size of lpBuff

Return Value

- `FMM_OK`: Command has been successfully performed.
- `FMM_NOT_OPEN`: The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM`: There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM`: There is no drive of iSlaveNo in the relevant port.

Remarks

Example

Refer to ‘FAS_Connect’ library.

See Also
FAS_IsSlaveExist

Check connection status of the drive

Syntax

```c
BOOL FAS_IsSlaveExist(
    BYTE nPortNo,
    BYTE iSlaveNo
);
```

Parameters

- `nPortNo` Port number of relevant drive
- `iSlaveNo` Slave number of relevant drive

Return Value

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

Remarks

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

Example

Refer to ‘FAS_Connect’ library.

See Also

FAS_Connect
**FAS_EnableLog**

To select the save function of communication error log file.

Syntax

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

Parameters

- `bEnable`
  Select output of Log.

Remarks

Select the Log output during Ezi-MOTION Plus-R DLL function used. This setup do not effect the other process or other program.

Log function start from 'FAS_Connect' function, the Log output is end when the 'FAS_Close' is executed.

Example

```c
#include "FAS_EziMOTIONPlusR.h"

void funcDisableLog()
{
    BYTE nPortNo = 1;

    FAS_EnableLog(FALSE);

    // Try to connect.
    if (FAS_Connect(nPortNo, 115200) == FALSE)
    {
        // connection fail.
        // can be different Port or different Baudrate.
        return;
    }

    // Connection close.
    FAS_Close(nPortNo);
}
```

See Also

- FAS_SetLogPath
**FAS_SetLogPath**

Setup the folder path of Log output files.

**Syntax**

```c
BOOL FAS_SetLogPath(LPCTSTR lpPath);
```

**Parameters**

- `lpPath`: Folder path Character string of Log output file.

**Return Value**

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

**Remarks**

This function have to be called before FAS_Connect library.

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

**Example**

```c
#include "FAS_EziMOTIONPlusR.h"

void funcEnableLog()
{
    BYTE nPortNo = 1; // COMM Port number

    // Log output.
    FAS_EnableLog(TRUE);

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

    // All Log output is stored in C:\Logs folder.

    // Try to connect.
    if (FAS_Connect(nPortNo, 115200) == FALSE)
    {
        // Connection fail.
        // cab be different Port or different Baudrate.
        return;
    }

    // Close connect.
    FAS_Close(nPortNo);
}
```

**See Also**

FAS_EnableLog
## 2-4. Parameter Control Function

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAS_SaveAllParameters</td>
<td>Save current status of parameters to the ROM:</td>
</tr>
<tr>
<td></td>
<td>Even after the drive is powered OFF, parameters related to operating speed, acceleration/deceleration time, and origin return need to be preserved.</td>
</tr>
<tr>
<td>FAS_SetParameter</td>
<td>Save designated parameter to the RAM:</td>
</tr>
<tr>
<td></td>
<td>Specific parameter is saved.</td>
</tr>
<tr>
<td>FAS_GetParameter</td>
<td>Read designated parameter from the RAM:</td>
</tr>
<tr>
<td></td>
<td>Specific parameter is read.</td>
</tr>
<tr>
<td>FAS_GetROMParameter</td>
<td>Read designated parameter from the ROM:</td>
</tr>
<tr>
<td></td>
<td>Specific parameter is read from the ROM.</td>
</tr>
</tbody>
</table>
FAS_SaveAllParameters

Save all edited parameters up to now and assigned I/O signals to the ROM area.

Syntax

```c
int FAS_SaveAllParameters(
    BYTE nPortNo,
    BYTE iSlaveNo
);
```

Parameters

- `nPortNo` Port number of relevant drive
- `iSlaveNo` Slave number of relevant drive

Return Value

- `FMM_OK` : Command has been successfully performed.
- `FMM_NOT_OPEN` : The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM` : There is no `nPort` in the connected ports.
- `FMM_INVALID_SLAVE_NUM` : There is no drive of `iSlaveNo` in the relevant port.

Remarks

Parameter values set to ‘FAS_SetIOAssignMap’ library as well as current parameter values are saved to the ROM.

Example

```c
#include "FAS_EziMOTIONPlusR.h"

void funcModifyParameter()
{
    BYTE nPortNo = 1; // COMM Port Number
    BYTE iSlaveNo = 0; // Slave No (0 ~ 15)
    long lParamVal;
    int nRtn;

    // Try to connect
    if (FAS_Connect(nPortNo, 115200) == FALSE)
    {
        // Connection failed.
        // The port is not connected or the baudrate may be wrong.
        return;
    }

    // Check Axis Start Speed Parameter.
    nRtn = FAS_GetParameter(nPortNo, iSlaveNo, STEP_AXISSTARTSPEED, &lParamVal);
    if (nRtn != FMM_OK)
    {
        // Command has not been performed properly.
        // Refer to ReturnCodes_Define.h.
        _ASSERT(FALSE);
    }
    else
    {
        // Parameter value saved in Ezi-STEP Plus-R.
        printf("Parameter [before] : Start Speed = %ld m/s", lParamVal);
    }
}
```
// Change Axis Start Speed parameter as 200 then read it again.
nRtn = FAS_SetParameter(nPortNo, iSlaveNo, STEP_AXISSTARTSPEED, 200);
ASSERT(nRtn == FMM_OK);  // You have to check if the command didn't execute correctly.

nRtn = FAS_GetParameter(nPortNo, iSlaveNo, STEP_AXISSTARTSPEED, &lParamVal);
ASSERT(nRtn == FMM_OK);
printf("Parameter [after] : Start Speed = %d\n", lParamVal);

// Check the value saved in the ROM.
nRtn = FAS_GetROMParameter(nPortNo, iSlaveNo, STEP_AXISSTARTSPEED, &lParamVal);
ASSERT(nRtn == FMM_OK);  // You have to check if the command didn't execute correctly.
printf("Parameter [ROM] : Start Speed = %d\n", lParamVal);

// Edit the parameter value then save it in the ROM.
nRtn = FAS_SetParameter(nPortNo, iSlaveNo, STEP_AXISSTARTSPEED, 100);
ASSERT(nRtn == FMM_OK);  // You have to check if the command didn't execute correctly.

nRtn = FAS_SaveAllParameters(nPortNo, iSlaveNo);
ASSERT(nRtn == FMM_OK);

// Disconnect.
FAS_Close(nPortNo);

See Also
FAS_GetROMParameter
**FAS_SetParameter**

Edit the relevant parameter value and then save it to the RAM.

Syntax

```c
int FAS_SetParameter(
    BYTE nPortNo,
    BYTE iSlaveNo,
    BYTE iParamNo,
    long lParamValue
);
```

Parameters

- `nPortNo`: Port number of relevant drive
- `iSlaveNo`: Slave number of relevant drive
- `iParamNo`: Parameter number to be edited
- `lParamValue`: Parameter value to be edited

Return Value

- `FMM_OK`: Command has been successfully performed.
- `FMM_NOT_OPEN`: The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM`: There is no `nPort` in the connected ports.
- `FMM_INVALID_SLAVE_NUM`: There is no drive of `iSlaveNo` in the relevant port.
- `FMM_INVALID_PARAMETER_NUM`: There is no parameter of designated `iParamNo`.

Remarks

The function operates for only one parameter designated. Parameters in the drive are saved to 2 memory areas. That is, when power is off, the ROM saves parameters permanently. When power is on, parameters in the ROM are copied to the DSP RAM and used. When the user changes parameters, it changes not parameters in the ROM but parameter in the RAM. This function is to set the parameter number designated from the RAM to the relevant value.

Example

Refer to ‘FAS_SaveAllParameter’ library.

See Also

- FAS_GetParameter
FAS_GetParameter

To call specific parameter value of the drive

Syntax

```c
int FAS_GetParameter(
    BYTE nPortNo,
    BYTE iSlaveNo,
    BYTE iParamNo,
    long* lParamValue
);
```

Parameters

- `nPortNo`: Port number of relevant drive
- `iSlaveNo`: Slave number of relevant drive
- `iParamNo`: Parameter number to be brought
- `lParamValue`: Parameter values

Return Value

- `FMM_OK`: Command has been successfully performed.
- `FMM_NOT_OPEN`: The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM`: There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM`: There is no drive of iSlaveNo in the relevant port.
- `FMM_INVALID_PARAMETER_NUM`: There is no parameter of designated iParamNo.

Remarks

The function operates for only one parameter designated. Parameters in the drive are saved to 2 memory areas. That is, when power is off, the ROM saves parameters permanently. When power is on, parameters in the ROM are copied to the DSP RAM and used. When the user changes parameters, it changes not parameters in the ROM but parameter in the RAM. This function reads the parameter number designated to the RAM.

Example

Refer to ‘FAS_SaveAllParameter’ library.

See Also

FAS_SetParameter
FAS_GetROMParameter

To call parameters saved in the ROM

Syntax

```c
int FAS_GetROMParameter(
    BYTE nPortNo,
    BYTE iSlaveNo,
    BYTE iParamNo,
    long* lROMParam
);
```

Parameters

- `nPortNo`: Port number of relevant drive
- `iSlaveNo`: Slave number of relevant drive
- `iParamNo`: Parameter number to be brought
- `lROMParam`: Parameter values saved in the ROM

Return Value

- `FMM_OK`: Command has been successfully performed.
- `FMM_NOT_OPEN`: The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM`: There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM`: There is no drive of iSlaveNo in the relevant port.
- `FMM_INVALID_PARAMETER_NUM`: There is no parameter of designated iParamNo.

Remarks

To call parameter values saved in the ROM

Even though this function runs, the value in the RAM is not changed. For this, run FAS_SetParameter.

Example

Refer to ‘FAS_SaveAllParameter’ library.

See Also

FAS_SaveAllParameters
### 2-5. Servo Control Function

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAS_STEP_ALARM_RESET</td>
<td>Release alarm of the drive generated alarm: Troubleshoot root cause of the alarm prior to use this function.</td>
</tr>
<tr>
<td>FAS_ALARM_TYPE</td>
<td>Read the Alarm type of the drive.</td>
</tr>
</tbody>
</table>
FAS_StepAlarmReset

To send AlarmReset command

Syntax

```c
int FAS_StepAlarmReset(
    BYTE nPortNo,
    BYTE iSlaveNo,
    BYTE bReset
);
```

Parameters

- **nPortNo**: Port number of relevant drive
- **iSlaveNo**: Slave number of relevant drive
- **bReset**: Reset command (1: reset, 0: reset release)

Return Value

- **FMM_OK**: Command has been successfully performed.
- **FMM_NOT_OPEN**: The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM**: There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM**: There is no drive of iSlaveNo in the relevant port.

Remarks

Before sending this command, troubleshoot root cause of the alarm.
For alarm cause, refer to ‘User Manual Text’.

Two times commands are needed for clearing the alarm status.
This command have to be executed sequentially ‘1’ and ‘0’ for the value
Of ‘bReset’. If you are execute only ‘1’ value, the motor will be ‘unlock’
Status.

Example

See Also
### 2-6. Control I/O Function

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAS_SetIOInput</td>
<td>To set the input signal level of the control input port:</td>
</tr>
<tr>
<td></td>
<td>Set input signal [ON] or [OFF] status.</td>
</tr>
<tr>
<td>FAS_GetIOInput</td>
<td>To read the current input signal status of the control input port:</td>
</tr>
<tr>
<td></td>
<td>The signal status returns by bit for each input signal.</td>
</tr>
<tr>
<td>FAS_SetIOOutput</td>
<td>To set the output signal level of the control input port:</td>
</tr>
<tr>
<td></td>
<td>Set output signal [ON] or [OFF] status.</td>
</tr>
<tr>
<td>FAS_GetIOOutput</td>
<td>To read the current input signal status of the control output port:</td>
</tr>
<tr>
<td></td>
<td>The signal status returns by bit for each output signal.</td>
</tr>
<tr>
<td>FAS_GetIOAssignMap</td>
<td>To read the pin of setting status of the CN1 port:</td>
</tr>
<tr>
<td></td>
<td>The setting status for each 9 variable signals returns by bit to the Input</td>
</tr>
<tr>
<td></td>
<td>and Output port.</td>
</tr>
<tr>
<td>FAS_SetIOAssignMap</td>
<td>To assign the control I/O signal to CN1 port pin and also set the signal</td>
</tr>
<tr>
<td></td>
<td>level:</td>
</tr>
<tr>
<td></td>
<td>Setting for each 9 variable signals is assigned to the Input and Output</td>
</tr>
<tr>
<td></td>
<td>port.</td>
</tr>
<tr>
<td>FAS_IOAssignMapReadROM</td>
<td>To load the pin of setting status of CN1 port from ROM area to RAM area.</td>
</tr>
</tbody>
</table>
FAS_SetIOInput

To set I/O input. For more information, refer to ‘1-1-5. Frame Type and Data Configuration’.

Syntax

```c
int FAS_SetIOInput(
    BYTE nPortNo,
    BYTE iSlaveNo,
    DWORD dwIOSetMask,
    DWORD dwIOCLRMask
);
```

Parameters

- `nPortNo`: Port number of relevant drive.
- `iSlaveNo`: Slave number of relevant drive.
- `dwIOSetMask`: Input bitmask value to be set
- `dwIOCLRMask`: Input bitmask value to be cleared

Return Value

- `FMM_OK`: Command has been successfully performed.
- `FMM_NOT_OPEN`: The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM`: There is no `nPort` in the connected ports.
- `FMM_INVALID_SLAVE_NUM`: There is no drive of `iSlaveNo` in the relevant port.

Remarks

Be careful that `dwIOSetMask` bit and `dwIOCLRMask` bit are not duplicated.

Example

```c
#include "FAS_EziMOTIONPlusR.h"

void funcIO()
{
    BYTE nPortNo = 1; // COMM Port Number
    BYTE iSlaveNo = 0; // Slave No (0 ~ 15)
    DWORD dwInput, dwOutput;
    int nRtn;

    // Try to connect
    if (FAS_Connect(nPortNo, 115200) == FALSE)
    {
        // Connection failed.
        // The port is not connected or the baudrate may be wrong.
        return;
    }

    // Check I/O input.
    nRtn = FAS_GetIOInput(nPortNo, iSlaveNo, &dwInput);
    _ASSERT(nRtn == FMM_OK);
    if (dwInput & STEP_IN_BITMASK_LIMITP)
    {
        // Limit + input is ON.
    }
}
if (dwInput & STEP_IN_BITMASK_USERIN0)
{
    // User Input 0 is ON.
}

// Turning ON 'Clear Position' and 'User Input 1' inputs and turning off 'Jog +' input.
nRtn = FAS_SetIOInput(nPortNo, iSlaveNo, STEP_IN_BITMASK_CLEARPOSITION | STEP_IN_BITMASK_USERIN1, STEP_IN_BITMASK_PJOG);
ASSERT(nRtn == FMM_OK);

// Check I/O output.
nRtn = FAS_GetIOOutput(nPortNo, iSlaveNo, &dwOutput);
ASSERT(nRtn == FMM_OK);
if (dwOutput & STEP_OUT_BITMASK_USEROUT0)
{
    // User Output 0 is ON.
}

// Turn off User Output 1 and 2 signals.
nRtn = FAS_SetIOOutput(nPortNo, iSlaveNo, 0, STEP_OUT_BITMASK_USEROUT1 | STEP_OUT_BITMASK_USEROUT2);
ASSERT(nRtn == FMM_OK);

// Disconnect.
FAS_Close(nPortNo);

See Also
FAS_GetIOInput
**FAS_GetIOInput**

To read I/O input values. For more information, refer to ‘1-1-5. Frame Type and Data Configuration’.

**Syntax**

```c
int FAS_GetIOInput(
    BYTE nPortNo,
    BYTE iSlaveNo,
    DWORD* dwIOInput
);
```

**Parameters**

- `nPortNo` Port number of relevant drive.
- `iSlaveNo` Slave number of relevant drive.
- `dwIOInput` Parameter pointer where input values will be saved.

**Return Value**

- **FMM_OK** : Command has been successfully performed.
- **FMM_NOT_OPEN** : The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM** : There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM** : There is no drive of iSlaveNo in the relevant port.

**Remarks**

There are 12 input pins in Ezi-STEP Plus-R. The user can select and use 9 input pins of them. This function can read the input port status as 32bit. All of them are insulated by a photocoupler. (Refer to the figure.)

![Diagram of input pins with photocouplers](image)

If voltage from an external input, is 24V at Port A, the input is recognized to 5V(High).

**Example**

Refer to ‘FAS_SetIOInput’ library.

**See Also**

FAS_SetIOInput
FAS_SetIOOutput

To set I/O output values. For more information, refer to ‘1-1-5. Frame Type and Data Configuration’.

Syntax

```c
int FAS_SetIOOutput(
    BYTE nPortNo,
    BYTE iSlaveNo,
    DWORD dwIOSetMask,
    DWORD dwIOCLRMask
);
```

Parameters

- **nPortNo**: Port number of relevant drive.
- **iSlaveNo**: Slave number of relevant drive.
- **dwIOSetMask**: Output bitmask value to be set (ON status)
- **dwIOCLRMask**: Output bitmask value to be cleared (OFF status)

Return Value

- **FMM_OK**: Command has been successfully performed.
- **FMM_NOT_OPEN**: The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM**: There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM**: There is no drive of iSlaveNo in the relevant port.

Remarks

There are 10 output pins in Ezi-STEP Plus-R. The user can select and use 9 output pins of them.

When output data is ‘1’, Port A becomes 0V. When it is ‘0’, Port A becomes +5V. Be careful that dwIOSetMask bit and dwIOCLRMask bit are not duplicated.

Example

Refer to FAS_SetIOInput.

See Also

FAS_GetIOOutput
**FAS_GetIOOutput**

To read I/O output values. For more information, refer to ‘1-1-5. Frame Type and Data Configuration’.

Syntax

```c
int FAS_GetIOOutput(
    BYTE nPortNo,
    BYTE iSlaveNo,
    DWORD* dwIOOutput
);
```

Parameters

- `nPortNo`: Port number of relevant drive.
- `iSlaveNo`: Slave number of relevant drive.
- `dwIOOutput`: Parameter pointer where the output value will be saved.

Return Value

- **FMM_OK**: Command has been successfully performed.
- **FMM_NOT_OPEN**: The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM**: There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM**: There is no drive of iSlaveNo in the relevant port.

Remarks

Example

Refer to ‘FAS_SetIOInput’ library

See Also

FAS_SetIOOutput
FAS_GetIOAssignMap

To read I/O Assign Map. For more information, refer to '1-1-5. Frame Type and Data Configuration'.

Syntax

```c
int FAS_GetIOAssignMap(
    BYTE nPortNo,
    BYTE iSlaveNo,
    BYTE iIPinNo,
    DWORD* dwIOLogicMask,
    BYTE* bLevel
);
```

Parameters

- `nPortNo`: Port number of relevant drive.
- `iSlaveNo`: Slave number of relevant drive.
- `iIPinNo`: I/O pin number to be read.
- `dwIOLogicMask`: Parameter pointer where the logic mask value assigned to a relevant pin will be saved.
- `bLevel`: Parameter pointer where the active level of relevant logic will be saved.

Return Value

- `FMM_OK`: Command has been successfully performed.
- `FMM_NOT_OPEN`: The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM`: There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM`: There is no drive of iSlaveNo in the relevant port.

Remarks

For `dwIOLogicMask`, refer to 'Motion_define.h'.

Example

```c
#include "FAS_EziMOTIONPlusR.h"

void funcIOAssign()
{
    BYTE nPortNo = 1; // COMM Port Number
    BYTE iSlaveNo = 0; // Slave No (0 ~ 15)
    BYTE iIPinNo;
    DWORD dwLogicMask;
    BYTE bLevel;
    BYTE i;
    int nRtn;

    // Try to connect
    if (FAS_Connect(nPortNo, 115200) == FALSE)
    {
        // Connection failed.
        // The port is not connected or the baudrate may be wrong.
        return;
    }

    // Check assigned information of input pin.
```
for (i=0; i</*Input Pin Count*//12; i++)
{
    nRtn = FAS_GetIOAssignMap(nPortNo, iSlaveNo, 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 E-Stop Logic (Low Active) to input pin 3.
iPinNo = 3: // 0 ~ 11 value is available (Caution : 0 ~ 2 is fixed.)
nRtn = FAS_SetIOAssignMap(nPortNo, iSlaveNo, iPinNo,
    STEP_IN_BITMASK_ESTOP, LEVEL_LOW_ACTIVE):
    _ASSERT(nRtn == FMM_OK):

// Check assign information of output pin.
for (i=0; i</*Output Pin Count*/; i++)
{
    nRtn = FAS_GetIOAssignMap(nPortNo, iSlaveNo, 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 is available (Caution : 0 is fixed to COMPOUT.)
nRtn = FAS_SetIOAssignMap(nPortNo, iSlaveNo, 12/*Input Pin Count*/ + iPinNo,
    STEP_OUT_BITMASK_ALARM, LEVEL_HIGH_ACTIVE):
    _ASSERT(nRtn == FMM_OK):

// Disconnect,
FAS_Close(nPortNo):
}

See Also
FAS_SetIOAssignMap
FAS_SetIOAssignMap

To set I/O Assign Map. For more information, refer to ‘1-1-5. Frame Type and Data Configuration’.

Syntax

```c
int FAS_SetIOAssignMap(
    BYTE nPortNo,
    BYTE iSlaveNo,
    BYTE iIOPinNo,
    DWORD dwIOLogicMask,
    BYTE bLevel
);
```

Parameters

- `nPortNo`  
  Port number of relevant drive.
- `iSlaveNo`  
  Slave number of relevant drive.
- `iIOPinNo`  
  I/O Pin number to be read
- `dwIOLogicMask`  
  Logic mask value to be assigned to the relevant pin
- `bLevel`  
  Active Level value of the relevant logic

Return Value

- `FMM_OK` : Command has been successfully performed.
- `FMM_NOT_OPEN` : The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM` : There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM` : There is no drive of iSlaveNo in the relevant port.
- `FMM_INVALID_PARAMETER_NUM` : Designated iIOPinNo or dwIOLogicMask value is out of range.

Remarks

To save current setting values to the ROM memory, ‘FAS_SaveAllParameters’ library should be run.

Example

Refer to ‘FAS_GSetIOAssignMap’ library

See Also

FAS_GetIOAssignMap
**FAS_I0AssignMapReadROM**

To load the status of CN1 assignment I/O setting status and signal level in ROM area.

**Syntax**

```c
int FAS_PosTableReadROM(
    BYTE nPortNo,
    BYTE iSlaveNo
);
```

**Parameters**

- **nPortNo**
  - Port number of relevant drive.
- **iSlaveNo**
  - Slave number of relevant drive.

**Return Value**

- **FMM_OK** : Command has been successfully performed.
- **FMM_NOT_OPEN** : The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM** : There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM** : There is no drive of iSlaveNo in the relevant port.

**Remarks**

**Example**

**See Also**

- FAS_GetIOAssignMap
2-7. Position Control Function

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAS_SetCommandPos</td>
<td>To set the command position value</td>
</tr>
<tr>
<td>FAS_SetActualPos</td>
<td>To set the current position to the actual position value</td>
</tr>
<tr>
<td>FAS_GetCommandPos</td>
<td>To read the current command position value</td>
</tr>
<tr>
<td>FAS_GetActualPos</td>
<td>To read the current actual position value</td>
</tr>
<tr>
<td>FAS_GetPosError</td>
<td>To read the difference between the actual position value and the command position value</td>
</tr>
<tr>
<td>FAS_GetActualVel</td>
<td>To read the actual running speed value while the motor is moving</td>
</tr>
<tr>
<td>FAS_ClearPosition</td>
<td>To set the command position and actual position value to ‘0’</td>
</tr>
</tbody>
</table>
**FAS_SetCommandPos**

To set the command position value of the motor

**Syntax**

```c
int FAS_SetCommandPos(
    BYTE nPortNo,
    BYTE iSlaveNo,
    long lCmdPos
);
```

**Parameters**

- **nPortNo**
  Port number of relevant drive.
- **iSlaveNo**
  Slave number of relevant drive.
- **lCmdPos**
  Command position value to be set.

**Return Value**

- **FMM_OK** : Command has been successfully performed.
- **FMM_NOT_OPEN** : The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM** : There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM** : There is no drive of iSlaveNo in the relevant port.

**Remarks**

The user sets the position command (pulse output counter) value. This function is generally used when the user sets the current position to coordinates that customer wants.

**Example**

```c
#include "FAS_EziMOTIONPlusR.h"

void funcClearPosition()
{
    BYTE nPortNo = 1;  // COMM Port Number
    BYTE iSlaveNo = 0;  // Slave No (0 ~ 15)
    int nRtn;

    // Try to connect
    if (FAS_Connect(nPortNo, 115200) == FALSE)
    {
        // Connection failed.
        // The port is not connected or the baudrate may be wrong.
        return;
    }

    // Initialize Command Position and Actual Position values to 0.
    nRtn = FAS_SetCommandPos(nPortNo, iSlaveNo, 0);
    _ASSERT(nRtn == FMM_OK);
    nRtn = FAS_SetActualPos(nPortNo, iSlaveNo, 0);
    _ASSERT(nRtn == FMM_OK);

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

**See Also**
FAS_SetActualPos
FAS_SetActualPos

To set the actual position value of the motor

Syntax

```c
int FAS_SetActualPos(
    BYTE nPortNo,
    BYTE iSlaveNo,
    long lActPos
);
```

Parameters

- `nPortNo`: Port number of relevant drive.
- `iSlaveNo`: Slave number of relevant drive.
- `lActPos`: Actual position value to be set.

Return Value

- `FMM_OK`: Command has been successfully performed.
- `FMM_NOT_OPEN`: The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM`: There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM`: There is no drive of iSlaveNo in the relevant port.

Remarks

Can be used when external encoder is connected.
The user sets the encoder feedback counter value to the value what customer wants.

Example

Refer to ‘FAS_GetActualPos’ library.

See Also

FAS_SetCommandPos
**FAS_GetCommandPos**

To read the command position of the current motor

**Syntax**

```c
int FAS_GetCommandPos(
    BYTE nPortNo,
    BYTE iSlaveNo,
    long* lCmdPos
);
```

**Parameters**

- **nPortNo**
  Port number of relevant drive
- **iSlaveNo**
  Slave number of relevant drive
- **lCmdPos**
  Parameter pointer where command position value will be saved

**Return Value**

- **FMM_OK** : Command has been successfully performed.
- **FMM_NOT_OPEN** : The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM** : There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM** : There is no drive of iSlaveNo in the relevant port.

**Remarks**

To read the position command (pulse output counter) value.

**Example**

```c
#include "FAS_EziMOTIONPlusR.h"

void funcDisplayStatus()
{
    BYTE nPortNo = 1; // COMM Port Number
    BYTE iSlaveNo = 0; // Slave No (0 ~ 15)
    long lValue;
    int nRtn;

    // Try to connect
    if (FAS_Connect(nPortNo, 115200) == FALSE)
    {
        // Connection failed.
        // The port is not connected or the baudrate may be wrong.
        return;
    }

    // Check position information of Ezi-STEP Plus-R.
    nRtn = FAS_GetCommandPos(nPortNo, iSlaveNo, &lValue);
    _ASSERT(nRtn == FMM_OK);
    printf("CMDPOS : %d\n", lValue);
    nRtn = FAS_GetActualVel(nPortNo, iSlaveNo, &lValue);
    _ASSERT(nRtn == FMM_OK);
    printf("ACTVEL : %d\n", lValue);

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

See Also
FAS_GetActualPos
**FAS_GetActualPos**

To read the actual position value of the motor

**Syntax**

```c
int FAS_GetActualPos(
    BYTE nPortNo,
    BYTE iSlaveNo,
    long* lActPos
);
```

**Parameters**

- `nPortNo`
  - Port number of relevant drive.
- `iSlaveNo`
  - Slave number of relevant drive.
- `lActPos`
  - Parameter pointer where the actual position value will be saved.

**Return Value**

- `FMM_OK` : Command has been successfully performed.
- `FMM_NOT_OPEN` : The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM` : There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM` : There is no drive of iSlaveNo in the relevant port.

**Remarks**

Can be used when external encoder is connected. When the user decides the motor position and checks its actual position, this function is generally used.

**Example**

Refer to ‘FAS_GetCommandPosition’ library.

**See Also**

FAS_GetCommandPos
FAS_GetPosError

To read the position error of the motor

Syntax

```c
int FAS_GetPosError(
    BYTE nPortNo,
    BYTE iSlaveNo,
    long* lPosErr
);
```

Parameters

- **nPortNo**
  - Port number of relevant drive.
- **iSlaveNo**
  - Slave number of relevant drive.
- **lPosErr**
  - Parameter pointer where the position error value will be saved

Return Value

- **FMM_OK** : Command has been successfully performed.
- **FMM_NOT_OPEN** : The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM** : There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM** : There is no drive of iSlaveNo in the relevant port.

Remarks

Can be used when external encoder is connected.

Example

Refer to ‘FAS_GetCommandPosition’ library.

See Also

- FAS_GetCommandPos
- FAS_GetActualPos
FAS_GetActualVel

To read the actual velocity of the motor

Syntax

```c
int FAS_GetActualVel(
    BYTE nPortNo,
    BYTE iSlaveNo,
    long* lActVel
);
```

Parameters

- `nPortNo`: Port number of relevant drive.
- `iSlaveNo`: Slave number of relevant drive.
- `lActVel`: Parameter pointer where the actual velocity value will be saved

Return Value

- `FMM_OK`: Command has been successfully performed.
- `FMM_NOT_OPEN`: The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM`: There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM`: There is no drive of iSlaveNo in the relevant port.

Remarks

Example

Refer to ‘FAS_GetCommandPosition’ library.

See Also
FAS_ClearPosition

To set the command position value and actual position value of the motor to ‘0’

Syntax

```c
int FAS_ClearPosition(
    BYTE nPortNo,
    BYTE iSlaveNo
);
```

Parameters

- `nPortNo`: Port number of relevant drive.
- `iSlaveNo`: Slave number of relevant drive.

Return Value

- **FMM_OK**: Command has been successfully performed.
- **FMM_NOT_OPEN**: The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM**: There is no `nPort` in the connected ports.
- **FMM_INVALID_SLAVE_NUM**: There is no drive of `iSlaveNo` in the relevant port.

Remarks

The user sets the position command (pulse output counter) value. This function is generally used when the user sets the current position to initial values.

Example

```c
#include "FAS_EziMOTIONPlusR.h"

void funcClearPosition()
{
    BYTE nPortNo = 1; // COMM Port Number
    BYTE iSlaveNo = 0; // Slave No (0 ~ 15)
    int nRtn;

    // Try to connect
    if (FAS_Connect(nPortNo, 115200) == FALSE)
    {
        // Connection failed.
        // The port is not connected or the baudrate may be wrong.
        return;
    }

    // Initialize Command Position and Actual Position values to 0.
    nRtn = FAS_ClearPosition(nPortNo, iSlaveNo);
    _ASSERT(nRtn == FMM_OK);

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

See Also

FAS_SetActualPos
## 2-8. Drive Status Control Function

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAS_GetIOAxisStatus</td>
<td>To read control I/O status, running status Flag value: The current input status value, the output setting status value, and the running status Flag value will be returned.</td>
</tr>
<tr>
<td>FAS_GetMotionStatus</td>
<td>To read the current running progress status and its PT number: The command position value, the actual position value, the speed value will be returned.</td>
</tr>
<tr>
<td>FAS_GetAllStatus</td>
<td>To read all status includes the current I/O status at one time: This function is to combine ‘FAS_GetIOAxisStatus’ function and ‘FAS_GetMotionStatus’ function.</td>
</tr>
<tr>
<td>FAS_GetAxisStatus</td>
<td>To read the running status Flag value of the relevant drive</td>
</tr>
</tbody>
</table>
FAS_GetI0AxisStatus

To read I/O Input and Output values of the relevant drive, and the motor Axis Status value

Syntax

```c
int FAS_GetI0AxisStatus(
    BYTE nPortNo,
    BYTE iSlaveNo,
    DWORD* dwInStatus,
    DWORD* dwOutStatus,
    DWORD* dwAxisStatus
);
```

Parameters

- `nPortNo` : Port number of relevant drive.
- `iSlaveNo` : Slave number of relevant drive.
- `dwInStatus` : Parameter pointer where the I/O input value will be saved.
- `dwOutStatus` : Parameter pointer where the I/O output value will be saved.
- `dwAxisStatus` : Parameter pointer where the axis status value of the relevant motor will be saved

Return Value

- `FMM_OK` : Command has been successfully performed.
- `FMM_NOT_OPEN` : The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM` : There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM` : There is no drive of iSlaveNo in the relevant port.

Remarks

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
**FAS_GetMotionStatus**

To read the motion status of current motor at one time

**Syntax**

```c
int FAS_GetMotionStatus(
    BYTE nPortNo,
    BYTE iSlaveNo,
    long* lCmdPos,
    long* lActPos,
    long* lPosErr,
    long* lActVel,
    WORD* wPosItemNo
);
```

**Parameters**

- `nPortNo` Port number of relevant drive.
- `iSlaveNo` Slave number of relevant drive.
- `lCmdPos` Parameter pointer where the command position value will be saved
- `lActPos` Parameter pointer where the actual position value will be saved.
- `lPosErr` Parameter pointer where the position error value will be saved
- `lActVel` Parameter pointer where the actual velocity value will be saved
- `wPosItemNo` Parameter pointer where current running item number in the Position Table will be saved

**Return Value**

- `FMM_OK` : Command has been successfully performed.
- `FMM_NOT_OPEN` : The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM` : There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM` : There is no drive of iSlaveNo in the relevant port.

**Remarks**

**Example**

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

**See Also**
FAS_GetAllStatus

To read I/O Input and Output values of the relevant drive, the motor Axis Status, the motor motion status at one time.

Syntax

```c
int FAS_GetAllStatus(
    BYTE nPortNo,
    BYTE iSlaveNo,
    DWORD* dwInStatus,
    DWORD* dwOutStatus,
    DWORD* dwAxisStatus,
    long* lCmdPos,
    long* lActPos,
    long* lPosErr,
    long* lActVel,
    WORD* wPosItemNo
);
```

Parameters

- `nPortNo`: Port number of relevant drive.
- `iSlaveNo`: Slave number of relevant drive.
- `dwInStatus`: Parameter pointer where the I/O input value will be saved.
- `dwOutStatus`: Parameter pointer where the I/O output value will be saved.
- `dwAxisStatus`: Parameter pointer where the axis status value of the relevant motor will be saved.
- `lCmdPos`: Parameter pointer where the command position value will be saved.
- `lActPos`: Parameter pointer where the actual position value will be saved.
- `lPosErr`: Parameter pointer where the position error value will be saved.
- `lActVel`: Parameter pointer where the actual velocity value will be saved.
- `wPosItemNo`: Parameter pointer where current running item number in the Position Table will be saved.

Return Value

- `FMM_OK`: Command has been successfully performed.
- `FMM_NOT_OPEN`: The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM`: There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM`: There is no drive of iSlaveNo in the relevant port.

Remarks

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also

FAS_GetAxisStatus
FAS_GetMotionStatus
**FAS_GetAxisStatus**

To read the motor Axis Status value. For status Flag, refer to ‘1-1-5. Frame Type and Data Configuration’.

**Syntax**

```c
int FAS_GetAxisStatus(
    BYTE nPortNo,
    BYTE iSlaveNo,
    DWORD* dwAxisStatus
);
```

**Parameters**

- **nPortNo**
  - Port number of relevant drive.
- **iSlaveNo**
  - Slave number of relevant drive.
- **dwAxisStatus**
  - Parameter pointer where the axis status value of the relevant motor

**Return Value**

- **FMM_OK**: Command has been successfully performed.
- **FMM_NOT_OPEN**: The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM**: There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM**: There is no drive of iSlaveNo in the relevant port.

**Remarks**

**Example**

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

**See Also**
### 2–9. Running Control Function

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAS_MoveStop</td>
<td>Stop the motor in running with deceleration.</td>
</tr>
<tr>
<td>FAS_EmergencyStop</td>
<td>Stop the motor in running immediately without deceleration.</td>
</tr>
<tr>
<td>FAS_MoveOriginSingleAxis</td>
<td>Start operation to return origin.</td>
</tr>
<tr>
<td>FAS_MoveSingleAxisAbsPos</td>
<td>The motor moves as much as the given absolute position value.</td>
</tr>
<tr>
<td>FAS_MoveSingleAxisIncPos</td>
<td>The motor moves as much as the given incremental position value.</td>
</tr>
<tr>
<td>FAS_MoveToLimit</td>
<td>The motor moves up to the position that the limit sensor is detected.</td>
</tr>
<tr>
<td>FAS_MoveVelocity</td>
<td>The motor moves to the given velocity and direction:</td>
</tr>
<tr>
<td></td>
<td>This function is available to Jog motion.</td>
</tr>
<tr>
<td>FAS_PositionAbsOverride</td>
<td>Changed the target absolute position value [pulse] of the motor in running.</td>
</tr>
<tr>
<td>FAS_PositionIncOverride</td>
<td>Changed the target incremental position value [pulse] of the motor in running.</td>
</tr>
<tr>
<td>FAS_VelocityOverride</td>
<td>Changed the running velocity value [pps] of the motor in running.</td>
</tr>
<tr>
<td>FAS_AllMoveStop</td>
<td>Stop all motors connected in same port with deceleration.</td>
</tr>
<tr>
<td>FAS_AllEmergencyStop</td>
<td>Stop all motors connected in same port immediately without deceleration.</td>
</tr>
<tr>
<td>FAS_AllMoveOriginSingleAxis</td>
<td>Start operation to return all motors in same port to origin position.</td>
</tr>
<tr>
<td>FAS_AllMoveSingleAxisAbsPos</td>
<td>All motors that connected in same port moves as much as the given absolute position value.</td>
</tr>
<tr>
<td>FAS_AllMoveSingleAxisIncPos</td>
<td>All motors that connected in same port moves as much as the given incremental position value.</td>
</tr>
<tr>
<td>FAS_MoveSingleAxisAbsPosEx</td>
<td>The motor moves as much as the given absolute position value with custom accel/decel time value.</td>
</tr>
<tr>
<td>FAS_MoveSingleAxisIncPosEx</td>
<td>The motor moves as much as the given incremental position value with custom accel/decel time value.</td>
</tr>
<tr>
<td>FAS_MoveVelocityEx</td>
<td>The motor moves to the given velocity and direction:</td>
</tr>
<tr>
<td></td>
<td>This function is available to Jog motion with custom accel/decel time value.</td>
</tr>
<tr>
<td>FAS_MovePause</td>
<td>The motor starts pause in running or the motor starts again in pause status.</td>
</tr>
</tbody>
</table>

- 77 -
FAS_MoveStop

To stop the motor

Syntax

```c
int FAS_MoveStop(
    BYTE nPortNo,
    BYTE iSlaveNo
);
```

Parameters

- `nPortNo`: Port number of relevant drive.
- `iSlaveNo`: Slave number of relevant drive.

Return Value

- `FMM_OK`: Command has been successfully performed.
- `FMM_NOT_OPEN`: The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM`: There is no `nPort` in the connected ports.
- `FMM_INVALID_SLAVE_NUM`: There is no drive of `iSlaveNo` in the relevant port.

Remarks

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
FAS_EmergencyStop

To stop the motor without deceleration

Syntax

```c
int FAS_EmergencyStop(
    BYTE nPortNo,
    BYTE iSlaveNo
);
```

Parameters

- **nPortNo**: Port number of relevant drive.
- **iSlaveNo**: Slave number of relevant drive.

Return Value

- **FMM_OK**: Command has been successfully performed.
- **FMM_NOT_OPEN**: The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM**: There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM**: There is no drive of iSlaveNo in the relevant port.

Remarks

This function does not include deceleration phase. So, the user must be careful so that the machine cannot be impacted.

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
FAS_MoveOriginSingleAxis

To search the origin of system. For more information, refer to 'User Manual_Text 9.3 Origin Return'.

Syntax

```c
int FAS_MoveOriginSingleAxis(
    BYTE nPortNo,
    BYTE iSlaveNo
);
```

Parameters

- **nPortNo**: Port number of relevant drive.
- **iSlaveNo**: Slave number of relevant drive.

Return Value

- **FMM_OK**: Command has been successfully performed.
- **FMM_NOT_OPEN**: The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM**: There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM**: There is no drive of iSlaveNo in the relevant port.

Remarks

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
**FAS_MoveSingleAxisAbsPos**

To move the motor to the absolute coordinate value

**Syntax**

```c
int FAS_MoveSingleAxisAbsPos(
    BYTE nPortNo,
    BYTE iSlaveNo,
    long lAbsPos,
    DWORD lVelocity
);
```

**Parameters**

- **nPortNo**
  Port number of relevant drive.
- **iSlaveNo**
  Slave number of relevant drive.
- **lAbsPos**
  Absolute coordinate where position to move
- **lVelocity**
  Velocity when the motor moves

**Return Value**

- **FMM_OK** : Command has been successfully performed.
- **FMM_NOT_OPEN** : The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM** : There is no `nPort` in the connected ports.
- **FMM_INVALID_SLAVE_NUM** : There is no drive of `iSlaveNo` in the relevant port.

**Remarks**

**Example**

```c
#include "FAS_EziMOTIONPlusR.h"

void funcMove()
{
    BYTE nPortNo = 1; // COMM Port Number
    BYTE iSlaveNo = 0; // Slave No (0 ~ 15)
    DWORD dwAxisStatus, dwInput;
    EZISTEP_AXISSTATUS stAxisStatus;
    long lAbsPos, lIncPos, lVelocity;
    int nRtn;

    // Try to connect
    if (FAS_Connect(nPortNo, 115200) == FALSE)
    {
        // Connection failed.
        // The port is not connected or the baudrate may be wrong.
        return;
    }

    // Check error status.
    nRtn = FAS_GetAxisStatus(nPortNo, iSlaveNo, &dwAxisStatus);
    _ASSERT(nRtn == FMM_OK);
    stAxisStatus.dwValue = dwAxisStatus;

    //if (dwAxisStatus & 0x00000001)
    if (stAxisStatus.FFLAG_ERRORALL)
    {
        // Do something
    }

    //if (dwAxisStatus & 0x00000001)
    if (stAxisStatus.FFLAG_ERRORALL)
    {
        // Do something
    }
}
```
FAS_StepAlarmReset(nPortNo, iSlaveNo):

// Check input status.
FMM_RESULT nRtn = FAS_GetInput(nPortNo, iSlaveNo, &dwInput);
='.$assert(nRtn == FMM_OK);

if (dwInput & (STEP_IN_BITMASK_STOP | STEP_IN_BITMASK_PAUSE | STEP_IN_BITMASK_ESTOP))
    FAS_SetInput(nPortNo, iSlaveNo, 0, STEP_IN_BITMASK_STOP | STEP_IN_BITMASK_PAUSE | STEP_IN_BITMASK_ESTOP);

// Increase the motor to 15000 pulse.
lIncPos = 15000;
lVelocity = 30000;
nRtn = FAS_MoveSingleAxisIncPos(nPortNo, iSlaveNo, lIncPos, lVelocity);
'.$assert(nRtn == FMM_OK);

// Stand by until motion command is completely finished.
do
{
    Sleep(1);
    nRtn = FAS_GetAxisStatus(nPortNo, iSlaveNo, &dwAxisStatus);
    '.$assert(nRtn == FMM_OK);
    stAxisStatus.dwValue = dwAxisStatus;
} while (stAxisStatus.FFLAG_MOTIONING);

// Move the motor to '0'.
lAbsPos = 0;
lVelocity = 20000;
nRtn = FAS_MoveSingleAxisAbsPos(nPortNo, iSlaveNo, lAbsPos, lVelocity);
'.$assert(nRtn == FMM_OK);

// Stand by until motion command is completely finished
    do
{
        Sleep(1);
        nRtn = FAS_GetAxisStatus(nPortNo, iSlaveNo, &dwAxisStatus);
        '.$assert(nRtn == FMM_OK);
        stAxisStatus.dwValue = dwAxisStatus;
    } while (stAxisStatus.FFLAG_MOTIONING);

// Disconnect.
FAS_Close(nPortNo);

See Also
FAS_MoveSingleAxisIncPos

To move the motor to the incremental coordinate value

Syntax

```c
int FAS_MoveSingleAxisIncPos(
    BYTE nPortNo,
    BYTE iSlaveNo,
    long lIncPos,
    DWORD lVelocity
);
```

Parameters

- `nPortNo` Port number of relevant drive.
- `iSlaveNo` Slave number of relevant drive.
- `lIncPos` Incremental coordinate where position to move
- `lVelocity` Velocity when the motor moves

Return Value

- `FMM_OK` : Command has been successfully performed.
- `FMM_NOT_OPEN` : The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM` : There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM` : There is no drive of iSlaveNo in the relevant port.

Remarks

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
FAS_MoveToLimit

To give the motor a command to search the limit sensor

Syntax

```c
int FAS_MoveToLimit(
    BYTE nPortNo,
    BYTE iSlaveNo,
    DWORD lVelocity,
    int iLimitDir
);
```

Parameters

- **nPortNo**: Port number of relevant drive.
- **iSlaveNo**: Slave number of relevant drive.
- **lVelocity**: Velocity when the motor moves.
- **iLimitDir**: Limit direction of the motor moves (0: -Limit, 1: +Limit)

Return Value

- **FMM_OK**: Command has been successfully performed.
- **FMM_NOT_OPEN**: The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM**: There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM**: There is no drive of iSlaveNo in the relevant port.

Remarks

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
FAS_MoveVelocity

To move the motor to the relevant direction and velocity. This function is available for Jog motion.

Syntax

```c
int FAS_MoveVelocity(
    BYTE nPortNo,
    BYTE iSlaveNo,
    DWORD lVelocity,
    int iVelDir
):
```

Parameters

- `nPortNo` Port number of relevant drive.
- `iSlaveNo` Slave number of relevant drive.
- `lVelocity` Velocity when the motor moves.
- `iVelDir` Direction when the motor moves (0: -Jog, 1: +Jog)

Return Value

- FMM_OK : Command has been successfully performed.
- FMM_NOT_OPEN : The drive has not been connected yet.
- FMM_INVALID_PORT_NUM : There is no nPort in the connected ports.
- FMM_INVALID_SLAVE_NUM : There is no drive of iSlaveNo in the relevant port.

Remarks

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
**FAS_PositionAbsOverride**

To change the absolute position value set while the motor moves to the absolute position

**Syntax**

```c
int FAS_PositionAbsOverride(
    BYTE nPortNo,
    BYTE iSlaveNo,
    long lOverridePos
);
```

**Parameters**

- `nPortNo`  
  Port number of relevant drive.
- `iSlaveNo`  
  Slave number of relevant drive.
- `lOverridePos`  
  Absolute coordinate position value to be changed

**Return Value**

- `FMM_OK` : Command has been successfully performed.
- `FMM_NOT_OPEN` : The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM` : There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM` : There is no drive of iSlaveNo in the relevant port.

**Remarks**

1) If the target position is set to the farther coordinate than the original target position while the motor moves under acceleration or constant velocity, the motor moves to the velocity pattern until then and stops at the target position.

![Graph](image1)

2) If the target position is changed while the motor is decelerated, it is again accelerated up to the constant velocity and then stops at the target position.

![Graph](image2)
3) If the changed target position is set to the closer coordinate than the original target position, the motor move to the changed target position.

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also

FAS_PositionIncOverride
**FAS_PositionIncOverride**

To change the incremental position value set while the motor moves to the incremental position

Syntax

```c
int FAS_PositionIncOverride(
    BYTE nPortNo,
    BYTE iSlaveNo,
    long lOverridePos
);
```

Parameters

- **nPortNo**: Port number of relevant drive.
- **iSlaveNo**: Slave number of relevant drive.
- **lOverridePos**: Incremental coordinate position value to be changed

Return Value

- **FMM_OK**: Command has been successfully performed.
- **FMM_NOT_OPEN**: The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM**: There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM**: There is no drive of iSlaveNo in the relevant port.

Remarks

Refer to ‘FAS_PositionAbsOverride’ library.

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also

FAS_PositionAbsOverride
# FAS_VelocityOverride

To change the velocity set while the motor moves

**Syntax**

```c
int FAS_VelocityOverride(
    BYTE nPortNo,
    BYTE iSlaveNo,
    DWORD lVelocity
);
```

**Parameters**

- **nPortNo**
  - Port number of relevant drive.
- **iSlaveNo**
  - Slave number of relevant drive.
- **lVelocity**
  - Velocity to be changed in [pps]

**Return Value**

- **FMM_OK** : Command has been successfully performed.
- **FMM_NOT_OPEN** : The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM** : There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM** : There is no drive of iSlaveNo in the relevant port.

**Remarks**

1) In case of \((\text{change speed}) < (\text{speed before change})\), the motor reaches to the change speed through acceleration/deceleration using a new velocity pattern.

2) In case of \((\text{change speed}) \geq (\text{speed before change})\), the motor reaches to the change speed through acceleration/deceleration without any new velocity pattern.

3) The motor reaches to the 'speed before change' without change of the velocity pattern and then it reaches to the 'change speed' by a new velocity pattern.

4) After acceleration/deceleration is finished, the motor reaches the change speed corresponding to the velocity pattern of the 'change speed'.

**Example**

Refer to 'FAS_MoveSingleAxisAbsPos' library.

**See Also**
FAS_AllMoveStop

To stop all motors that connected in same port.

Syntax

```c
int FAS_AllMoveStop(
    BYTE nPortNo
);
```

Parameters

- `nPortNo` Port number of relevant drive.

Return Value

No response

Remarks

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
**FAS_AllEmergencyStop**

To stop all motors that connected in same port without deceleration

Syntax

```c
int FAS_AllEmergencyStop(
    BYTE nPortNo
);
```

Parameters

- `nPortNo`
  - Port number of relevant drive.

Return Value

- No response

Remarks

This function does not include deceleration phase. So, the user must be careful so that the machine cannot be impacted.

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
FAS_AllMoveOriginSingleAxis

To search the origin of system for all motor those are connected in same port. For more information, refer to ‘User Manual_Text 9.3 Origin Return’.

Syntax

```c
int FAS_AllMoveOriginSingleAxis(
    BYTE nPortNo
);
```

Parameters

- `nPortNo`
  - Port number of relevant drive.

Return Value

- No response

Remarks

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
**FAS_AllMoveSingleAxisAbsPos**

To move all motors that connected in same port to the absolute coordinate

Syntax

```c
int FAS_AllMoveSingleAxisAbsPos(
    BYTE nPortNo,
    long lAbsPos,
    DWORD lVelocity
);
```

Parameters

- `nPortNo` Port number of relevant drive.
- `lAbsPos` Absolute coordinate of position to move
- `lVelocity` Velocity when the motor moves

Return Value

- No response

Remarks

Example

Refer to 'FAS_MoveSingleAxisAbsPos' Library.

See Also
FAS_AllMoveSingleAxisIncPos

To move all motors that connected in same port to the incremental coordinate value

Syntax

```c
int FAS_AllMoveSingleAxisIncPos(  
    BYTE nPortNo,  
    long lIncPos,  
    DWORD lVelocity
);
```

Parameters

- `nPortNo`  
  Port number of relevant drive.
- `lIncPos`  
  Incremental coordinate of position to move
- `lVelocity`  
  Velocity when the motor moves

Return Value

- No response

Remarks

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
FAS_MoveSingleAxisAbsPosEx

To move the motor to the absolute coordinate

Syntax

```c
int FAS_MoveSingleAxisAbsPosEx(
    BYTE nPortNo,
    BYTE iSlaveNo,
    long lAbsPos,
    DWORD lVelocity,
    MOTION_OPTION_EX* lpExOption
);
```

Parameters

- **nPortNo**
  - Port number of relevant drive.
- **iSlaveNo**
  - Slave number of relevant drive.
- **lAbsPos**
  - Absolute coordinate of position to move
- **lVelocity**
  - Velocity when the motor moves
- **lpExOption**
  - Custom option.

Return Value

- **FMM_OK** : Command has been normally performed.
- **FMM_NOT_OPEN** : The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM** : There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM** : There is no drive of iSlaveNo in the relevant port.

Remarks

Refer to MOTION_OPTION_EX struct.

Example

```c
#include "FAS_EziMOTIONPlusR.h"

void funcMoveEx()
{
    BYTE nPortNo = 1;  // COMM Port Number
    BYTE iSlaveNo = 0; // Slave No (0 ~ 15)
    DWORD dwAxisStatus, dwInput;
    EZISTEP_AXISSTATUS stAxisStatus;
    long lAbsPos, llIncPos, lVelocity;
    MOTION_OPTION_EX opt = {0};
    int nRtn;

    // Try to connect
    if (FAS_Connect(nPortNo, 115200) == FALSE)
    {
        // Connection failed.
        // The port number may be wrong, or incorrect Baudrate.
        return;
    }

    // Moving motor with different acc/dec time
    llIncPos = 15000;
    lVelocity = 30000;
```
opt.flagOption.BIT_USE_CUSTOMACCEL = 1;
opt.flagOption.BIT_USE_CUSTOMDECEL = 1;

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

nRtn = FAS_MoveSingleAxisIncPosEx(nPortNo, iSlaveNo, lIncPos, lVelocity, &opt);
ASSERT(nRtn == FMM_OK);

// Waiting until motioning is done.
do{
    Sleep(1);
    nRtn = FAS_GetAxisStatus(nPortNo, iSlaveNo, &dwAxisStatus);
    ASSERT(nRtn == FMM_OK);
    stAxisStatus.dwValue = dwAxisStatus;
}while (stAxisStatus.FFLAG_MOTIONING);

// Moving motor to position 0.
lAbsPos = 0;
lVelocity = 20000;
nRtn = FAS_MoveSingleAxisAbsPos(nPortNo, iSlaveNo, lAbsPos, lVelocity);
ASSERT(nRtn == FMM_OK);

// Waiting until motioning is done.
do{
    Sleep(1);
    nRtn = FAS_GetAxisStatus(nPortNo, iSlaveNo, &dwAxisStatus);
    ASSERT(nRtn == FMM_OK);
    stAxisStatus.dwValue = dwAxisStatus;
}while (stAxisStatus.FFLAG_MOTIONING);

// Disconnect.
FAS_Close(nPortNo);

See Also
FAS_MoveSingleAxisIncPosEx

To move the motor to the Incremental coordinate

Syntax

```c
int FAS_MoveSingleAxisIncPosEx(
    BYTE nPortNo,
    BYTE iSlaveNo,
    long lIncPos,
    DWORD lVelocity,
    MOTION_OPTION_EX* lpExOption
);
```

Parameters

- **nPortNo**
  - Port number of relevant drive.
- **iSlaveNo**
  - Slave number of relevant drive.
- **lIncPos**
  - Incremental coordinate of position to move
- **lVelocity**
  - Velocity when the motor moves
- **lpExOption**
  - Custom option.

Return Value

- **FMM_OK** : Command has been normally performed.
- **FMM_NOT_OPEN** : The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM** : There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM** : There is no drive of iSlaveNo in the relevant port.

Remarks

Example

See Also
FAS_MoveVelocityEx

To move the motor to the relevant direction and velocity. This function is also available for Jog motion.

Syntax

```c
int FAS_MoveSingleAxisIncPosEx(
    BYTE nPortNo,
    BYTE iSlaveNo,
    DWORD lVelocity,
    int iVelDir,
    VELOCITY_OPTION_EX* lpExOption
);
```

Parameters

- `nPortNo` Port number of relevant drive.
- `iSlaveNo` Slave number of relevant drive.
- `lVelocity` Velocity when the motor moves
- `iVelDir` Direction which the motor moves (0: -Jog, 1: +Jog)
- `lpExOption` Custom option.

Return Value

- `FMM_OK` : Command has been normally performed.
- `FMM_NOT_OPEN` : The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM` : There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM` : There is no drive of iSlaveNo in the relevant port.

Remarks

Refer to VELOCITY_OPTION_EX struct.

Example

```c
#include "FAS_EziMOTIONPlusR.h"

void funcMoveVelocityEx()
{
    BYTE nPortNo = 1; // COMM Port Number
    BYTE iSlaveNo = 0; // Slave No (0 ~ 15)
    long lVelocity;
    VELOCITY_OPTION_EX opt = {0};
    int nRtn;

    // Try to connect
    if (FAS_Connect(nPortNo, 115200) == FALSE)
    {
        // Connection failed.
        printf("The port number may be wrong, or incorrect Baudrate.",
            return;
        }

    // Moving motor with different acc/dec time : FAS_MoveSingleAxisIncPosEx
    lVelocity = 30000;
    opt.flagOption.BIT_USE_CUSTOMACCDEC = 1;
    opt.wCustomAccDecTime = 300;
}
```
nRtn = FAS_MoveVelocityEx(nPortNo, iSlaveNo, lVelocity, DIR_INC, &opt);
ASSERT(nRtn == FMM_OK);
Sleep(5000);
FAS_MoveStop(nPortNo, iSlaveNo);
}

See Also
## 2-10. Position Table Control Function

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAS_PosTableReadItem</td>
<td>To read items of RAM area in the specific position table</td>
</tr>
<tr>
<td>FAS_PosTableWriteItem</td>
<td>To save specific position table to RAM area</td>
</tr>
<tr>
<td>FAS_PosTableWriteROM</td>
<td>To save all of position table values to ROM area: Total 256 PT values are saved.</td>
</tr>
<tr>
<td>FAS_PosTableReadROM</td>
<td>To read position table values in ROM area: Total 256 PT values are read.</td>
</tr>
<tr>
<td>FAS_PosTableRunItem</td>
<td>The motor starts to run from the designated position table in sequence.</td>
</tr>
<tr>
<td>FAS_PosTableReadOneItem</td>
<td>To read items of RAM area in the specific one item of position table</td>
</tr>
<tr>
<td>FAS_PosTableWriteOneItem</td>
<td>To save specific item of specific position table to RAM area</td>
</tr>
</tbody>
</table>
**FAS_PosTableReadItem**

To read a specific item in the position table

**Syntax**

```c
int FAS_PosTableReadItem(
    BYTE nPortNo,
    BYTE iSlaveNo,
    WORD wItemNo,
    LPITEM_NODE lpItem
);
```

**Parameters**

- **nPortNo**: Port number of relevant drive.
- **iSlaveNo**: Slave number of relevant drive.
- **wItemNo**: Item number to be read
- **lpItem**: Item structure pointer where item value is saved

**Return Value**

- **FMM_OK**: Command has been successfully performed.
- **FMM_NOT_OPEN**: The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM**: There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM**: There is no drive of iSlaveNo in the relevant port.
- **FMM_INVALID_PARAMETER_NUM**: wItemNo is out of range.

**Remarks**

**Example**

```c
#include "FAS_EziMOTIONPlusR.h"

void funcPosTable()
{
    BYTE nPortNo = 1; // COMM Port Number
    BYTE iSlaveNo = 0; // Slave No (0 ~ 15)
    WORD wItemNo:
    ITEM_NODE nodeItem:
    int nRtn:

    // Try to connect
    if (FAS_Connect(nPortNo, 115200) == FALSE)
    {
        // Connection failed.
        // The port is not connected or the baudrate may be wrong.
        return;
    }

    // Read No.20 Position table value and edit the position value.
    wItemNo = 20:
    nRtn = FAS_PosTableReadItem(nPortNo, iSlaveNo, wItemNo, &nodeItem):
    _ASSERT(nRtn == FMM_OK):

    nodeItem.iPosition = 260000; // Change the position value to 260000.
    nodeItem.wBranch = 23; // Set next command to 23.
```
nodeItem.wContinuous = 1; // Next command should be connected without deceleration.

nRtn = FAS_PosTableWriteItem(nPortNo, iSlaveNo, wItemNo, &nodeItem);
ASSERT(nRtn == FMM_OK);

// Call the value in the ROM regardless of edited position table data.
 nRtn = FAS_PosTableReadROM(nPortNo, iSlaveNo);
ASSERT(nRtn == FMM_OK);

// Save edited position table data in the ROM.
 nRtn = FAS_PosTableWriteROM(nPortNo, iSlaveNo);
ASSERT(nRtn == FMM_OK);

// Disconnect.
FAS_Close(nPortNo);

} See Also
FAS_PosTableWriteItem
**FAS_PosTableWriteItem**

To edit specific items in the position table

Syntax

```c
int FAS_PosTableWriteItem(
    BYTE nPortNo,
    BYTE iSlaveNo,
    WORD wItemNo,
    LPITEM_NODE lpItem
);
```

Parameters

- `nPortNo` Port number of relevant drive.
- `iSlaveNo` Slave number of relevant drive.
- `wItemNo` Item number to be edited
- `lpItem` Item structure pointer to be edited

Return Value

- `FMM_OK` : Command has been successfully performed.
- `FMM_NOT_OPEN` : The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM` : There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM` : There is no drive of iSlaveNo in the relevant port.
- `FMC_POSTABLE_ERROR` : An error occurs while position table is being written.
- `FMM_INVALID_PARAMETER_NUM` : wItemNo is out of range.

Remarks

Position Table data is saved to RAM / ROM area. This function activates to save data to RAM area. When power is off, data is deleted.

Example

See Also

- `FAS_PosTableReadItem`
FAS_PosTableWriteROM

To save all current position table items to ROM area

Syntax

```c
int FAS_PosTableWriteROM(
    BYTE nPortNo,
    BYTE iSlaveNo
);
```

Parameters

- `nPortNo` Port number of relevant drive.
- `iSlaveNo` Slave number of relevant drive.

Return Value

- `FMM_OK` : Command has been successfully performed.
- `FMM_NOT_OPEN` : The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM` : There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM` : There is no drive of iSlaveNo in the relevant port.
- `FMC_POSTABLE_ERROR` : An error occurs while position table is being saved.

Remarks

Position table data is saved to RAM / ROM area. This function activates to save data to ROM area. Even though power is off, data is preserved.

Example

See Also

- FAS_PosTableReadROM
**FAS_PosTableReadROM**

To read position table items being saved in ROM area

Syntax

```c
int FAS_PosTableReadROM(
    BYTE nPortNo,
    BYTE iSlaveNo
);
```

Parameters

- `nPortNo` Port number of relevant drive.
- `iSlaveNo` Slave number of relevant drive.

Return Value

- **FMM_OK** : Command has been successfully performed.
- **FMM_NOT_OPEN** : The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM** : There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM** : There is no drive of iSlaveNo in the relevant port.
- **FMC_POSTABLE_ERROR** : An error occurs while position table is being read.

Remarks

Example

See Also

- FAS_PosTableWriteROM
FAS_PosTableRunItem

To perform command from a specific item in the position table

Syntax

```c
int FAS_PosTableRunItem(
    BYTE nPortNo,
    BYTE iSlaveNo,
    WORD wItemNo
);
```

Parameters

- `nPortNo` Port number of relevant drive.
- `iSlaveNo` Slave number of relevant drive.
- `wItemNo` Item number to start motion

Return Value

- **FMM_OK**: Command has been successfully performed.
- **FMM_NOT_OPEN**: The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM**: There is no `nPort` in the connected ports.
- **FMM_INVALID_SLAVE_NUM**: There is no drive of `iSlaveNo` in the relevant port.
- **FMM_INVALID_PARAMETER_NUM**: `wItemNo` is out of range.

Remarks

Example

See Also

- FAS_GetAllStatus
- FAS_MoveStop
- FAS_EmergencyStop
FAS_PosTableReadOneItem

To read specific item in the specific position table

Syntax

```c
int FAS_PosTableReadOneItem(
    BYTE nPortNo,
    BYTE iSlaveNo,
    WORD wItemNo,
    WORD wOffset,
    long* lPosItemVal
);
```

Parameters

- `nPortNo` Port number of relevant drive.
- `iSlaveNo` Slave number of relevant drive.
- `wItemNo` Item number to be read
- `wOffset` Offset value which will be read from PT items. (Refer to '1-2-6. Position Table Item')
- `lPosItemVal` Parameter pointer where PT item data value will be saved

Return Value

- `FMM_OK` : Command has been successfully performed.
- `FMM_NOT_OPEN` : The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM` : There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM` : There is no drive of iSlaveNo in the relevant port.
- `FMM_INVALID_PARAMETER_NUM` : wItemNo is out of range.

Remarks

Example

See Also

- FAS_PosTableWriteOneItem
FAS_PosTableWriteOneItem

To edit specific item in the specific position table

Syntax

```c
int FAS_PosTableWriteOneItem(
    BYTE nPortNo,
    BYTE iSlaveNo,
    WORD wItemNo,
    WORD wOffset,
    long lPosItemVal
);
```

Parameters

- `nPortNo`: Port number of relevant drive.
- `iSlaveNo`: Slave number of relevant drive.
- `wItemNo`: Item number to be edited.
- `wOffset`: Offset value which will be saved from PT items. (Refer to ‘1-2-6. Position Table Item’)
- `lPosItemVal`: PT item data value to be set

Return Value

- `FMM_OK`: Command has been successfully performed.
- `FMM_NOT_OPEN`: The drive has not been connected yet.
- `FMM_INVALID_PORT_NUM`: There is no nPort in the connected ports.
- `FMM_INVALID_SLAVE_NUM`: There is no drive of iSlaveNo in the relevant port.
- `FMC_POSTABLE_ERROR`: An error occurs while position table is being written.
- `FMM_INVALID_PARAMETER_NUM`: wItemNo is out of range.

Remarks

Example

See Also

FAS_PosTableReadOneItem
2—11. Other Control Function

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAS_TriggerOutput_RunA</td>
<td>To Start/Stop command for ‘Compare Out’ signal</td>
</tr>
<tr>
<td>FAS_TriggerOutput_Status</td>
<td>To check if the trigger output pulse is working or not.</td>
</tr>
</tbody>
</table>
**FAS_TriggerOutput_RunA**

To start/stop the digital output signal (Compare Out pin) when reaching the specific Target position.

Syntax

```c
int FAS_TriggerOutput_RunA(
    BYTE nPortNo,
    BYTE iSlaveNo,
    BOOL bStartTrigger,
    long lStartPos,
    DWORD dwPeriod,
    DWORD dwPulseTime,
);
```

Parameters

- **nPortNo**
  - Port number of relevant drive.
- **iSlaveNo**
  - Slave number of relevant drive.
- **bStartTrigger**
  - Output start/stop command (1:start, 0:stop)
- **lStartPos**
  - Output start position [pulse]
- **dwPeriod**
  - Period of output signal [pulse]
- **dwPulseTime**
  - Width of output signal [msec]

Return Value

- **FMM_OK** : Command has been normally performed.
- **FMM_NOT_OPEN** : The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM** : There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM** : There is no drive of iSlaveNo in the relevant port.
- **FMM_INVALID_PARAMETER_NUM** : wItemNo is out of range.

Remarks

Example

See Also

FAS_TriggerOutput_Status
**FAS_TriggerOutput_Status**

To check if the trigger output is working or not.

Syntax

```c
int FAS_TriggerOutput_Status(
    BYTE nPortNo,
    BYTE iSlaveNo,
    BYTE* bTriggerStatus
);
```

Parameters

- **nPortNo**
  - Port number of relevant drive.
- **iSlaveNo**
  - Slave number of relevant drive.
- **bTriggerStatus**
  - Current status of signal output.

Return Value

- **FMM_OK** : Command has been normally performed.
- **FMM_NOT_OPEN** : The drive has not been connected yet.
- **FMM_INVALID_PORT_NUM** : There is no nPort in the connected ports.
- **FMM_INVALID_SLAVE_NUM** : There is no drive of iSlaveNo in the relevant port.

Remarks

Example

See Also

- FAS_TriggerOutput_RunA
3. Protocol for PLC Program

Next window activates when you click the icon in User Program(GUI) installed folder.

Next test procedure will help you to understand the protocol programming.

(1) Servo ON/OFF command purpose of command

* In case of Ezi-STEP Plus-R : Jump to next step(‘(2) Motion Command’), because the motor is ready to move status after Power ON.

The header and tail information is needed for protocol programming. Additionally Frame Data (Slave ID, Frame type, Data and CRC) is also needed in each one of protocol with header and tail.

1) Select ‘Comm Port’ number and ‘Baudrate’, and click ‘Connect’ button.

2) Header: Click ‘Header’ and you can see ‘[0xAA][0xCC]’ on ‘Send Buffer’ window.

3) Slave ID: Insert your connected slave number (above example is ‘0’) and click ‘SlaveNo’.

4) Frame type: Select ‘Frame type’.

You can find next table information in ‘1-2-1. Frame Type and Data Configuration’ on UserManual(Ezi-SERVO Plus-R)_Communication Function about Servo ON/OFF command.

<table>
<thead>
<tr>
<th>Frame type</th>
<th>DLL Library name</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>42 (0x2A)</td>
<td>FAS_ServoEnable</td>
<td>Setting the Servo ON/OFF status. Sending: 1 byte</td>
</tr>
</tbody>
</table>

Insert ‘42’ in ① area and click ‘1 byte’ because the size of Frame Type is 1 byte.

5) Data: To make Servo ON status, the data is ‘1’ so insert ‘1’ in ④ area and click ‘1 byte’.

6) CRC: Click ‘CRC’ and the automatically calculated result value (2 bytes) is displayed on ‘Send Buffer’ window.
7) Tail: click ‘Tail’ and you can see ‘[0xAA][0xEE]’ on ‘Send Buffer’ window.
8) Finally click ‘Send’ button to send command characters to Ezi-SERVO Plus-R.
You can check the motor torque and LED flash for Servo ON status.
9) After sending command, you can check the answering information from Ezi-SERVO
Plus-R on ‘Buffer Received’ window.

(2) Motion command purpose of command

1) Header
2) Slave No.
4) Data (Position value): insert ‘10000’ and click ‘4 byte’.
5) Data (Running speed): insert ‘5000’ and click ‘4 byte’.
6) CRC
7) Tail
8) Send: When parameter sets as ‘default’ value, motor rotates as one revolution. ‘53’
   command is incremental move command so once click ‘Send’, motor will rotate again as
   same distance.

(3) PLC Programming
In ‘Protocol test GUI’ automatically calculate the ‘Byte stuffing’ and ‘CRC’ data.
For protocol programming in PLC, you have to add the function of ‘Byte stuffing’ and
‘CRC’ calculation.
For ‘Byte stuffing’ refer to ‘1-1-2. RS-485 Communication Protocol’ and for ‘CRC’ refer
to ‘1-1-3. CRC Calculation Example’ on UserManual(Ezi-STEP Plus-R) Communication
Function.