User Manual
Communication Function_Ver6

( Rev.08.05.11 )
- Table of Contents -

1. Communication Protocols........................................................................................................ 6

   1-1. Communication Functions.................................................................................................. 6
      1-1-1. Communication Specifications................................................................. 6
      1-1-2. RS-485 Communication Protocol(Ver6)......................................................... 6
      1-1-3. CRC Calculation Example .............................................................................. 7
      1-1-4. Response Frame Structure and Communication Error(Ver6) ......................... 10

   1-2. Structure of Frame type(Ver6)....................................................................................... 11
      1-2-1. Frame type and Data Configuration.......................................................... 11
      1-2-2. Parameter Lists ......................................................................................... 23
      1-2-3. Bit setup of Output pin ............................................................................. 24
      1-2-4. Bit setup of Input pin .............................................................................. 24
      1-2-5. Bit setup of Status Flag ............................................................................ 25
      1-2-6. Position Table Item ................................................................................. 27
      1-2-7. Information of Motors ............................................................................. 27

   1-3. Program Method............................................................................................................ 28

2. Library for PC Program(Ver6).......................................................................................... 29

   2-1. Library Configuration................................................................................................. 29

   2-2. Communication Status Window.................................................................................. 30

   2-3. Drive Link Function.................................................................................................... 35
       FAS_Connect........................................................................................................ 36
       FAS_Close.......................................................................................................... 38
       FAS_GetSlaveInfo.............................................................................................. 39
       FAS_GetMotorInfo............................................................................................ 40
       FAS_IsSlaveExist............................................................................................. 41
## 2-4. Parameter Control Function

- FAS_SaveAllParameters ................................................................. 43
- FAS_SetParameter ................................................................. 45
- FAS_GetParameter ................................................................. 46
- FAS_GetROMParameter ............................................................. 47

## 2-5. Servo Control Function

- FAS_ServoEnable ................................................................. 49
- FAS_ServoAlarmReset ............................................................. 51

## 2-6. Control I/O Function

- 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

- 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

- FAS_GetIOAxisStatus .............................................................. 73
- FAS_GetMotionStatus .............................................................. 74
- FAS_GetAllStatus ................................................................. 75
- FAS_GetAxisStatus ................................................................. 76
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_MoveLinearAbsPos / FAS_MoveLinearIncPos ..................................... 95

2—10. Position Table Control Function ...................................................... 96
   FAS_PosTableReadItem .......................................................................... 97
   FAS_PosTableWriteItem ......................................................................... 99
   FAS_PosTableWriteROM ....................................................................... 100
   FAS_PosTableReadROM ....................................................................... 101
   FAS_PosTableRunItem ......................................................................... 102
   FAS_PosTableReadOneItem .................................................................. 103
   FAS_PosTableWriteOneItem .................................................................. 104

2—11. Other Control Function ................................................................. 105
   FAS_TriggerOutput_RunA ....................................................................... 106
   FAS_TriggerOutput_Status ..................................................................... 107
   FAS_MovePush ....................................................................................... 108
   FAS_GetPushStatus ............................................................................... 109
3. Protocol for PLC Program ........................................................................................................ 110
1. Communication Protocols

1-1. Communication Functions

Ezi-SERVO Plus-R can control up to 16 axes by multidrop link at RS-485(two-wire).

Pay attention that when Windows goes into standby or power-save mode, serial communication is basically disconnected. When the system is recovered from standby mode, it should be connected again with serial communication. This is also applicable to the library provided.

1-1-1. Communication Specifications

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

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

There are 2 kinds of program version for SERVO 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-SERVO

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>4-252 bytes</td>
<td>0xAA 0xEE</td>
</tr>
</tbody>
</table>

① 0xAA : Delimited byte
② 0xAA 0xCC : Displays that the Frame locates in header.
③ 0xAA 0xEE : Displays that the Frame locates in tail.
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 displays that an error has occurred.

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>
<tr>
<td>Low byte</td>
<td>High byte</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

① Slave ID : Dive module number (0-15) connected to the PC communication port.
② Frame type : To 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 that an error occurs during communication. ‘0xA001’ of a polynomial factor in CRC16(Cyclic Redundancy Check) is used. Or ‘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, 0X0240, 0X03D0, 0X04A1, 0X05F1, 0X0660,
    0X07D0, 0X08C1, 0X09B1, 0X0AB0, 0X0BD1, 0X0CE0, 0X0DF1, 0X0EE0,
    0X10F0, 0X1181, 0X1271, 0X13A0, 0X14D0, 0X15C1, 0X16B1, 0X1740,
    0X18D0, 0X19C1, 0X1AB1, 0X1BD0, 0X1CE1, 0X1DF0, 0X1EE0, 0X1FF1,
    0X2080, 0X2170, 0X22A0, 0X23D1, 0X24C1, 0X25F0, 0X2660, 0X27D0,
    0X28C1, 0X29B1, 0X2AB0, 0X2BD1, 0X2CE0, 0X2DF1, 0X2EE0, 0X2FF1,
    0X3080, 0X3170, 0X32A0, 0X33D1, 0X34C1, 0X35F0, 0X3660, 0X37D0,
    0X38C1, 0X39B1, 0X3AB0, 0X3BD1, 0X3CE0, 0X3DF1, 0X3EE0, 0X3FF1,
    0X4080, 0X4170, 0X42A0, 0X43D1, 0X44C1, 0X45F0, 0X4660, 0X47D0,
    0X48C1, 0X49B1, 0X4AB0, 0X4BD1, 0X4CE0, 0X4DF1, 0X4EE0, 0X4FF1,
    0X5080, 0X5170, 0X52A0, 0X53D1, 0X54C1, 0X55F0, 0X5660, 0X57D0,
    0X58C1, 0X59B1, 0X5AB0, 0X5BD1, 0X5CE0, 0X5DF1, 0X5EE0, 0X5FF1,
    0X6080, 0X6170, 0X62A0, 0X63D1, 0X64C1, 0X65F0, 0X6660, 0X67D0,
    0X68C1, 0X69B1, 0X6AB0, 0X6BD1, 0X6CE0, 0X6DF1, 0X6EE0, 0X6FF1,
    0X7080, 0X7170, 0X72A0, 0X73D1, 0X74C1, 0X75F0, 0X7660, 0X77D0,
    0X78C1, 0X79B1, 0X7AB0, 0X7BD1, 0X7CE0, 0X7DF1, 0X7EE0, 0X7FF1,
    0X8080, 0X8170, 0X82A0, 0X83D1, 0X84C1, 0X85F0, 0X8660, 0X87D0,
    0X88C1, 0X89B1, 0X8AB0, 0X8BD1, 0X8CE0, 0X8DF1, 0X8EE0, 0X8FF1,
    0X9080, 0X9170, 0X92A0, 0X93D1, 0X94C1, 0X95F0, 0X9660, 0X97D0,
    0X98C1, 0X99B1, 0X9AB0, 0X9BD1, 0X9CE0, 0X9DF1, 0X9EE0, 0X9FF1,
    0xA080, 0xA170, 0xA2A0, 0xA3D1, 0xA4C1, 0xA5F0, 0xA660, 0xA7D0,
    0xA8C1, 0xA9B1, 0xAA00, 0xABD1, 0xAC60, 0xADF1, 0xAE00, 0xAF10,
    0xB020, 0xB110, 0xB240, 0xB3D1, 0xB4C1, 0xB5F0, 0xB660, 0xB7D0,
    0xB8C1, 0xB9B1, 0xBA00, 0xBBD1, 0xBC60, 0xBD91, 0xBE00, 0xBF10,
    0xC020, 0xC110, 0xC240, 0xC3D1, 0xC4C1, 0xC590, 0xC600, 0xC710,
    0xC820, 0xC910, 0xCA00, 0xCB91, 0xCD00, 0xCE10, 0xCF20, 0xDF10,
    0xE020, 0xE110, 0xE240, 0xE3D1, 0xE4C1, 0xE590, 0xE600, 0xE710,
    0xE820, 0xE910, 0xEA00, 0xEB91, 0xEC00, 0xED10, 0xEF20, 0xDF10,
    0xF020, 0xF110, 0xF240, 0xF3D1, 0xF4C1, 0xF590, 0xF600, 0xF710,
    0xF820, 0xF910, 0xFA00, 0xFB91, 0xFC00, 0xFD10, 0xFE20, 0xFF10,
};
```
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;
}

unsigned short CalcCRCbyAlgorithm(unsigned char* pDataBuffer, unsigned long usDataLen)
{
    // Use the Modbus algorithm as detailed in the Watlow comms guide
    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);
    }
    return wCrc;
}
/* Loop through all 8 data bits */

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 same. 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, it should be recognized as the error status.)
2. Frame type : Same to sending Frame.
   (When this is not same to sending data, it should be recognized 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 added to display the communication status (error / normal).

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>0x80</td>
<td>128</td>
<td>Frame Type Error : Responded Frame type cannot be recognized.</td>
</tr>
<tr>
<td>0x81</td>
<td>129</td>
<td>Data error. ROM data read/write error : Data value responded is without the given range.</td>
</tr>
<tr>
<td>0x82</td>
<td>130</td>
<td>Received Frame Error : Frame data received is out of this specification.</td>
</tr>
</tbody>
</table>
| 0x85      | 133          | Running Command Failure : The user has tried to execute new running commands in wrong condition as follows.
   1) currently motor is running
   2) currently motor is stopping
   3) currently Servo is OFF status
   4) try to Z-pulse Origin without encoder |
| 0x86      | 134          | RESET Failure : The user has tried to execute new running commands in wrong condition as follows.
   1) While the servo is ON
   2) Already RESET in ON by external input signal |
| 0x87      | 135          | Servo ON Failure ① : While an alarm occurs, the user has tried to execute Servo ON command. |
| 0x88      | 136          | Servo ON Failure ② : While Emergency Stop occurs, the user has tried to execute Servo ON command. |
| 0x89      | 137          | Servo ON Failure ③ : ‘ServoON’ signal is assigned to input pin already. Servo ON/OFF can execute by external input signal only. |
| 0xAA      | 170          | CRC Error : Frame data received is out of CRC format. In this case, DLL Library of sending side automatically try to send 1 more time. |

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 displays the content and configuration of data by Frame type.

<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 1 byte 0-246 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status Slave type ACII string with NULL byte (strlen() + 1 bytes)</td>
</tr>
<tr>
<td>0x05 (5)</td>
<td>FAS_GetMotorInfo</td>
<td>Connected motor type and maker 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 1 byte 0-246 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status Motor type ACII string with NULL byte (strlen() + 1 bytes)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>♦ Motor type: refer to '1-1-7.Information of Motors'</td>
</tr>
<tr>
<td>0x10 (16)</td>
<td>FAS_SaveAllParameters</td>
<td>Current setting parameters &amp; assign of IO signals are saved in the ROM of the drive. Even though the drive is powered off, saving these must be possible. Values set at 'FAS_SetParameter' &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 Communication status</td>
</tr>
<tr>
<td>0x11 (17)</td>
<td>FAS_GetRomParameter</td>
<td>Specific parameter values in the ROM are read.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 byte Parameter number (0-31)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 5 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 byte 4 bytes Communication status Parameter value</td>
</tr>
</tbody>
</table>

Refer to '1-2-2.Parameter List'
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Sending Data</th>
<th>Response Data</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x12 (18)</td>
<td>FAS_SetParameter</td>
<td>Specific parameter values are saved to the RAM.</td>
<td>1 byte</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending : 5 bytes</td>
<td>4 bytes</td>
<td>Communication status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parameter number (0-31) Parameter value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x13 (19)</td>
<td>FAS_GetParameter</td>
<td>Specific parameter values in the RAM are read</td>
<td>1 byte</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending : 1 byte</td>
<td>Parameter number (0-31)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response : 5 bytes</td>
<td>4 bytes</td>
<td>Communication status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Parameter value</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0x20 (32)</td>
<td>FAS_SetIOOutput</td>
<td>Output signal level of the control output port is set.</td>
<td>4 bytes</td>
<td>4 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending : 8 bytes</td>
<td>I/O set mask value I/O clear mask value</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>When specific bit of the set mask is ‘1’ the relevant output port signal is set to [ON].</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>When specific bit of the clear mask is ‘1’ the relevant output port signal is set to [OFF].</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response : 1 byte</td>
<td>Communication status</td>
<td></td>
</tr>
<tr>
<td>0x21 (33)</td>
<td>FAS_SetIOInput</td>
<td>Input signal level of the control input port is set.</td>
<td>4 bytes</td>
<td>4 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending : 8 bytes</td>
<td>I/O set mask value I/O clear mask value</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>When specific bit of the set mask is ‘1’ the relevant input port signal is set to [ON].</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>When specific bit of the clear mask is ‘1’ the relevant input port signal is set to [OFF].</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response : 1 byte</td>
<td>Communication status</td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td>Command</td>
<td>Description</td>
<td>Sending</td>
<td>Response</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------</td>
<td>------------------------------------------------------------------------------------------------</td>
<td>---------</td>
<td>----------</td>
</tr>
<tr>
<td>0x22</td>
<td>FAS_GetIOInput</td>
<td>Current input signal status of the control input port is read.</td>
<td>0 byte</td>
<td>5 byte</td>
</tr>
<tr>
<td>0x23</td>
<td>FAS_GetIOOutput</td>
<td>Current output signal status of the control output port is read.</td>
<td>0 byte</td>
<td>5 byte</td>
</tr>
</tbody>
</table>
| 0x24  | FAS_SetIOAssignMap | To assign control I/O signals to the pin of CN1 port and set the signal level. By running ‘FAS_SaveAllParameters’, you can save the setting value to the ROM. | 6 bytes | 1 byte   | 1 byte 4 bytes 1 byte | I/O number I/O pin masking data Setting level | ♦ 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 |
| 0x25  | FAS_GetIOAssignMap | Pin setting status of CN1 port is read from RAM area.                                             | 1 byte  | 6 bytes  | 1 byte 4 bytes 1 byte | Communication status | I/O number: ‘0-11’ corresponds to ‘Limit+, Limit-, Org, IN1,…, IN9’ respectively, and ‘12-22’ corresponds to ‘COMP, OUT1,…, OUT9’ respectively.  
For more information, refer to ‘0x24’ Frame type. |
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x26</td>
<td>FAS_IOAssignMapReadROM</td>
<td>Pin setting status of CN1 port is loaded to RAM from ROM area.</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>1 byte</td>
</tr>
<tr>
<td></td>
<td>Communication status</td>
<td>Command performing status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0 : complete, values except 0: error)</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>4 bytes</td>
</tr>
<tr>
<td></td>
<td>4 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Output start/stop</td>
<td>Pulse start position</td>
</tr>
<tr>
<td></td>
<td>(1:start 0:stop)</td>
<td>Pulse period</td>
</tr>
<tr>
<td></td>
<td>4 byte</td>
<td>1 bytes</td>
</tr>
<tr>
<td></td>
<td>Pulse width</td>
<td>Output pin number</td>
</tr>
<tr>
<td></td>
<td>[msec]</td>
<td>(fix to 0)</td>
</tr>
<tr>
<td></td>
<td>4 byte</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td>Communication status</td>
<td>Command performing status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0 : complete, values except 0: error)</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 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>1 bytes</td>
</tr>
<tr>
<td></td>
<td>Communication status</td>
<td>Status (1:output ON, 0 :output OFF)</td>
</tr>
<tr>
<td>0x2A</td>
<td>FAS_ServoEnable</td>
<td>Servo 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></td>
</tr>
<tr>
<td></td>
<td>0:OFF, 1:ON</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>0x2B</td>
<td>FAS_ServoAlarmReset</td>
<td>Servo alarm status is reset.</td>
</tr>
<tr>
<td></td>
<td>Sending : 0 byte</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>Request</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------------------</td>
<td>---------</td>
</tr>
<tr>
<td>0x2E</td>
<td>FAS_ServoAlarntype</td>
<td>To request the Alarm type</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>Communication status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>♦ Alarm type: No alarm (0) OverCurrent(1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OverSpeed(2) StepOut(3) OverLoad(4)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OverTemperature(5) BackEMF(6) MotorConnect(7)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>EncoderConnect(8) MotorPower(9) Inposition(10)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SystemHalt(11) ROMdevicel(12) OverInputVoltage(14)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Position Overflow(15)</td>
</tr>
<tr>
<td>0x31</td>
<td>FAS_MoveStop</td>
<td>To request to stop running the motor</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>Communication status</td>
</tr>
<tr>
<td>0x32</td>
<td>FAS_EmergencyStop</td>
<td>To request the running motor to stop emergently</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>Communication status</td>
</tr>
<tr>
<td>0x33</td>
<td>FAS_MoveOriginSingleAxis</td>
<td>To request the motor to return to the origin at the current setting parameter condition</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>Communication status</td>
</tr>
<tr>
<td>0x34</td>
<td>FAS_MoveSingleAxisAbsPos</td>
<td>To request the motor to move its position as much as the absolute value[pulse]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 8 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 bytes 4 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Absolute position value Running speed [pps]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 1 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status</td>
</tr>
<tr>
<td>Code</td>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0x35</td>
<td>FAS_MoveSingleAxisIncPos</td>
<td>To request the motor to move its position as much as the incremental value[pulse]</td>
</tr>
<tr>
<td></td>
<td>Sending: 8 bytes</td>
<td>4 bytes Incremental position value 4 bytes Running speed [pps]</td>
</tr>
<tr>
<td></td>
<td>Response: 1 byte</td>
<td>1 byte Communication status</td>
</tr>
<tr>
<td>0x36</td>
<td>FAS_MoveToLimit</td>
<td>To request the motor to start limit motion at the current setting parameter condition</td>
</tr>
<tr>
<td></td>
<td>Sending: 5 bytes</td>
<td>4 bytes Running speed [pps] 1 byte Running direction (0: -Limit  1: +Limit)</td>
</tr>
<tr>
<td></td>
<td>Response: 1 byte</td>
<td>1 byte Communication status</td>
</tr>
<tr>
<td>0x37</td>
<td>FAS_MoveVelocity</td>
<td>To request the motor to start jog motion at the current setting parameter condition</td>
</tr>
<tr>
<td></td>
<td>Sending: 5 bytes</td>
<td>4 bytes Running speed [pps] 1 byte Running direction (0: -Jog  1: +Jog)</td>
</tr>
<tr>
<td></td>
<td>Response: 1 byte</td>
<td>1 byte Communication status</td>
</tr>
<tr>
<td>0x38</td>
<td>FAS_PositionAbsOverride</td>
<td>To request the motor to change the target absolute position value[pulse] while it is in running.</td>
</tr>
<tr>
<td></td>
<td>Sending: 4 bytes</td>
<td>4 bytes Changed command position value[pulse]</td>
</tr>
<tr>
<td></td>
<td>Response: 1 byte</td>
<td>1 byte Communication status</td>
</tr>
<tr>
<td>0x39</td>
<td>FAS_PositionIncOverride</td>
<td>To request the motor to change the target incremental position value[pulse] while it is in running.</td>
</tr>
<tr>
<td></td>
<td>Sending: 4 bytes</td>
<td>4 bytes Changed command position value[pulse]</td>
</tr>
<tr>
<td></td>
<td>Response: 1 byte</td>
<td>1 byte Communication status</td>
</tr>
<tr>
<td>0x3A</td>
<td>FAS_VelocityOverride</td>
<td>To request the motor to change the running speed value[pps] while it is in running.</td>
</tr>
<tr>
<td></td>
<td>Sending: 4 bytes</td>
<td>4 bytes Changed running speed [pps]</td>
</tr>
<tr>
<td></td>
<td>Response: 1 byte</td>
<td>1 byte Communication status</td>
</tr>
<tr>
<td>Code</td>
<td>Command Name</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------</td>
<td>---------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>0x3B</td>
<td>FAS_AllMoveStop</td>
<td>To request stop for all motor that connected in same port.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Slave number must be ‘99’)</td>
</tr>
<tr>
<td>0x3C</td>
<td>FAS_AllEmergencyStop</td>
<td>To request emergency stop for all motor that connected in same port.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Slave number must be ‘99’)</td>
</tr>
<tr>
<td>0x3D</td>
<td>FAS_AllMoveOriginSingleAxis</td>
<td>To request return to the origin at the current setting parameter condition for all motors that connected in same port.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(Slave number must be ‘99’)</td>
</tr>
<tr>
<td>0x3E</td>
<td>FAS_AllSingleAxisAbsPos</td>
<td>To request move its position as much as the absolute value[pulse] for all motors that connected in same port.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 bytes 4 bytes</td>
</tr>
<tr>
<td>0x3F</td>
<td>FAS_AllSingleAxisIncPos</td>
<td>To request move its position as much as the incremental value[pulse] for all motors that connected in same port.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4 bytes 4 bytes</td>
</tr>
</tbody>
</table>

FAS_MoveLinearAbsPos: To request Linear Interpolation move its position as much as the absolute value[pulse] for more than 2 motors that connected in same port. Refer to 「2. Library for PC program」.

FAS_MoveLinearIncPos: To request Linear Interpolation move its position as much as the incremental value[pulse] for more than 2 motors that connected in same port. Refer to 「2. Library for PC program」.
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Functionality</th>
<th>Sending</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x40</td>
<td>FAS_GetAxisStatus</td>
<td>To request the Flag value of displaying the running status</td>
<td>0 byte</td>
<td>5 bytes</td>
</tr>
<tr>
<td>0x41</td>
<td>FAS_GetIOAxisStatus</td>
<td>To request the I/O status and the running Flag status. (Frame type 0x22, 0x23, and 0x40 are packed.)</td>
<td>0 byte</td>
<td>13 bytes</td>
</tr>
<tr>
<td>0x42</td>
<td>FAS_GetMotionStatus</td>
<td>To request the current running progress status and its PT number. (Frame type 0x51, 0x53, 0x54, and 0x55 are packed.)</td>
<td>0 byte</td>
<td>21 bytes</td>
</tr>
<tr>
<td>0x43</td>
<td>FAS_GetAllStatus</td>
<td>To request all data including the current running status. (Frame type 0x41, and 0x42 are packed.)</td>
<td>0 byte</td>
<td>33 bytes</td>
</tr>
</tbody>
</table>

### FAS_GetAxisStatus

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

### FAS_GetIOAxisStatus

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

### FAS_GetMotionStatus

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication status</td>
<td>1 byte</td>
</tr>
<tr>
<td>Command position value</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Actual Position value</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Position Difference value</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Running speed value</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Current running value</td>
<td>4 bytes</td>
</tr>
<tr>
<td>PT number</td>
<td>4 bytes</td>
</tr>
</tbody>
</table>

### FAS_GetAllStatus

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command position value</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Actual Position value</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Position Difference value</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Running speed value</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Current running value</td>
<td>4 bytes</td>
</tr>
<tr>
<td>PT number</td>
<td>4 bytes</td>
</tr>
<tr>
<td>Code</td>
<td>Function</td>
</tr>
<tr>
<td>------</td>
<td>----------</td>
</tr>
<tr>
<td>0x50 (80)</td>
<td>FAS_SetCommandPos</td>
</tr>
<tr>
<td>0x51 (81)</td>
<td>FAS_GetCommandPos</td>
</tr>
<tr>
<td>0x52 (82)</td>
<td>FAS_SetActualPos</td>
</tr>
<tr>
<td>0x53 (83)</td>
<td>FAS_GetActualPos</td>
</tr>
<tr>
<td>0x54 (84)</td>
<td>FAS_GetPosError</td>
</tr>
</tbody>
</table>

By this value, the user can check the current running status (how much inposition is tracked).
<table>
<thead>
<tr>
<th>Code</th>
<th>Description</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x55</td>
<td>FAS_GetActualVel</td>
<td>To request the current running speed value [pps]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 0 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 5 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status</td>
</tr>
<tr>
<td>0x56</td>
<td>FAS_ClearPosition</td>
<td>Ezi-SERVO Plus-R is the closed loop control drive and so the command</td>
</tr>
<tr>
<td></td>
<td></td>
<td>position value is continuously controlled while the motor is in running.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>The user sets the command position and actual position value to ‘0’</td>
</tr>
<tr>
<td></td>
<td></td>
<td>before it starts to operate and then can check how the command position</td>
</tr>
<tr>
<td></td>
<td></td>
<td>value is changed.</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></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status</td>
</tr>
<tr>
<td>0x60</td>
<td>FAS_PosTableReadItem</td>
<td>To read PT 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></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Readable PT number (0-255)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 65 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For items by each PT, refer to 「1-2-6. Position Table Item」.</td>
</tr>
<tr>
<td>0x61</td>
<td>FAS_PosTableWriteItem</td>
<td>To save PT values to the RAM of the drive.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 66 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>PT number (0-255)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For items by each PT, refer to 「1-2-6. Position Table Item」.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 2 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(values except 0 : complete, 0: error)</td>
</tr>
<tr>
<td>0x62</td>
<td>FAS_PosTableReadROM</td>
<td>To read all PT values (256 ea) in the ROM of the drive</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Sending: 0 byte</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Response: 2 bytes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Communication status</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(0 : complete, values except 0: error)</td>
</tr>
<tr>
<td>Code</td>
<td>Command Name</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>0x63</td>
<td>FAS_PosTableWriteROM</td>
<td>To save all PT value (256 ea) to the ROM of the drive.</td>
</tr>
<tr>
<td></td>
<td>Sending: 0 byte</td>
<td>Response: 2 bytes</td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>1 byte</td>
</tr>
<tr>
<td></td>
<td>Communication status</td>
<td>Command performing status (0: complete, values except 0: error)</td>
</tr>
<tr>
<td>0x64</td>
<td>FAS_PosTableRunItem</td>
<td>To start the position table operation from the designated PT number</td>
</tr>
<tr>
<td></td>
<td>Sending: 2 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>PT Number (0-255)</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>0x6A</td>
<td>FAS_PosTableReadOneItem</td>
<td>To read one of PT values in the RAM of the drive.</td>
</tr>
<tr>
<td></td>
<td>Sending: 4 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 byte</td>
<td>2 byte</td>
</tr>
<tr>
<td></td>
<td>PT Number (0-255)</td>
<td>Offset value (0-40)</td>
</tr>
<tr>
<td></td>
<td>Response: 5 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td>4 byte</td>
</tr>
<tr>
<td></td>
<td>Communication status</td>
<td>Relevant one of PT value</td>
</tr>
<tr>
<td>0x6B</td>
<td>FAS_PosTableWriteOneItem</td>
<td>To save one of PT values to the RAM of the drive.</td>
</tr>
<tr>
<td></td>
<td>Sending: 8 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2 byte</td>
<td>2 byte</td>
</tr>
<tr>
<td></td>
<td>PT Number (0-255)</td>
<td>Offset value (0-40)</td>
</tr>
<tr>
<td></td>
<td>Response: 2 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1 byte</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Communication status</td>
<td>Command performing status (values except 0: complete, 0: error)</td>
</tr>
<tr>
<td>0x78</td>
<td>FAS_MovePush</td>
<td>To request push motion (maintain specified motor torque) command</td>
</tr>
<tr>
<td></td>
<td>Sending: 26 bytes</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4 byte</td>
<td>4 bytes</td>
</tr>
<tr>
<td></td>
<td>Normal Start speed</td>
<td>Normal Move speed</td>
</tr>
<tr>
<td></td>
<td>2 byte</td>
<td>4 bytes</td>
</tr>
<tr>
<td></td>
<td>Push torque ratio</td>
<td>Push Move speed</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>
</tbody>
</table>
| Frame Type '0x65' ~ '0x69', '0x90' ~ '0x92' are allotted for internal use.
| PT Number : 0-255 for Ezi-SERVO-PR
| 0-63 for Ezi-SERVO-PR-MI

<table>
<thead>
<tr>
<th>0x79 (121)</th>
<th>FAS_GetPushStatus</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>To request the current push motion status.</td>
</tr>
<tr>
<td></td>
<td>Sending: 0 byte</td>
</tr>
<tr>
<td></td>
<td>Response: 2 byte</td>
</tr>
<tr>
<td>1 byte</td>
<td>1 bytes</td>
</tr>
<tr>
<td>Communication status</td>
<td>Push motion status</td>
</tr>
<tr>
<td></td>
<td>(0: normal Servo ON</td>
</tr>
<tr>
<td></td>
<td>1: push motioning but the work is not detected</td>
</tr>
<tr>
<td></td>
<td>2: work detected and torque is maintained</td>
</tr>
</tbody>
</table>
### 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>9</td>
<td>9</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>0</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>0</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>Position Loop Gain</td>
<td></td>
<td>0</td>
<td>15</td>
<td>4</td>
</tr>
<tr>
<td>26</td>
<td>Inpos Value</td>
<td></td>
<td>0</td>
<td>15</td>
<td>0</td>
</tr>
<tr>
<td>27</td>
<td>Pos Tracking Limit</td>
<td>[pulse]</td>
<td>1</td>
<td>+134,217,727</td>
<td>5,000</td>
</tr>
<tr>
<td>28</td>
<td>Motion Dir</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>29</td>
<td>Limit Sensor Dir</td>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>Org Torque Ratio</td>
<td>[%]</td>
<td>10</td>
<td>100</td>
<td>50</td>
</tr>
<tr>
<td>31</td>
<td>Pos. Error Overflow Limit</td>
<td>[pulse]</td>
<td>1</td>
<td>+134,217,727</td>
<td>5,000</td>
</tr>
</tbody>
</table>
1–2–3. Bit setup of Output pin

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

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Relevant Bit</th>
<th>Signal Name</th>
<th>Relevant Bit</th>
<th>Signal Name</th>
<th>Relevant Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compare Out</td>
<td>0x00000001</td>
<td>Origin Search OK</td>
<td>0x00000010</td>
<td>User Output 1</td>
<td>0x00010000</td>
</tr>
<tr>
<td>Imposition</td>
<td>0x00000002</td>
<td>ServoReady</td>
<td>0x00000200</td>
<td>User Output 2</td>
<td>0x00020000</td>
</tr>
<tr>
<td>Alarm</td>
<td>0x00000004</td>
<td>reserved</td>
<td>0x00001000</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 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>0x00100000</td>
<td>0x00000000</td>
</tr>
</tbody>
</table>

【Example 2】 Sending data to turn OFF the User Output 5 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>0x00100000</td>
</tr>
</tbody>
</table>

1–2–4. Bit setup of Input pin

This displays the detailed description for 0x21 Frame type. This command is applicable to 32 signals in the control input port. The user can use signals for test 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</th>
<th>Signal Name</th>
<th>Relevant Bit</th>
<th>Signal Name</th>
<th>Relevant Bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Limit+</td>
<td>0x00000001</td>
<td>PT A4</td>
<td>0x00000100</td>
<td>AlarmReset</td>
<td>0x00001000</td>
</tr>
<tr>
<td>Limit-</td>
<td>0x00000002</td>
<td>PT A5</td>
<td>0x00000200</td>
<td>ServoON</td>
<td>0x00002000</td>
</tr>
<tr>
<td>Origin</td>
<td>0x00000004</td>
<td>PT A6</td>
<td>0x00000400</td>
<td>Pause</td>
<td>0x00004000</td>
</tr>
<tr>
<td>Clear Position</td>
<td>0x00000008</td>
<td>PT A7</td>
<td>0x00000800</td>
<td>Org Search</td>
<td>0x00080000</td>
</tr>
<tr>
<td>PT A0</td>
<td>0x00000010</td>
<td>PT Start</td>
<td>0x00010000</td>
<td>Teaching</td>
<td>0x00100000</td>
</tr>
<tr>
<td>PT A1</td>
<td>0x00000020</td>
<td>Stop</td>
<td>0x00020000</td>
<td>E-stop</td>
<td>0x00200000</td>
</tr>
<tr>
<td>PT A2</td>
<td>0x00000040</td>
<td>Jog+</td>
<td>0x00040000</td>
<td>JPT input 0</td>
<td>0x00400000</td>
</tr>
<tr>
<td>PT A3</td>
<td>0x00000080</td>
<td>Jog-</td>
<td>0x00080000</td>
<td>JPT input 1</td>
<td>0x00800000</td>
</tr>
</tbody>
</table>

【Example 1】 Sending data to turn ON the Pause port
### 1-2-5. Bit setup of Status Flag

Refer to ‘motion_define.h’ of include files.
It can be applied under the Firmware version of 06.03.043.8

<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 error occurs.</td>
<td>0x00000001</td>
</tr>
<tr>
<td>FFLAG_HWPOSILMT</td>
<td>‘+’ direction limit sensor turns ON.</td>
<td>0x00000002</td>
</tr>
<tr>
<td>FFLAG_HWNNEGALMT</td>
<td>‘-’ direction limit sensor turns ON.</td>
<td>0x00000004</td>
</tr>
<tr>
<td>FFLAG_SWPOGILMT</td>
<td>‘+’ direction program limit is exceeded.</td>
<td>0x00000008</td>
</tr>
<tr>
<td>FFLAG_SWNNEGALMT</td>
<td>‘-’ direction program limit is exceeded.</td>
<td>0x00000010</td>
</tr>
<tr>
<td>reserved</td>
<td></td>
<td>0x00000020</td>
</tr>
<tr>
<td>FFLAG_POSCTOVER</td>
<td>Position error is higher than ‘Pos Tracking Limit’ value during position command run.</td>
<td>0x00000040</td>
</tr>
<tr>
<td>FFLAG_ERRSERVOALARM</td>
<td>One or more error of Servo alarm(8 ea) occurs.</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>FFLAG_ERROVERLOAD</td>
<td>Load exceeding the max torque of the motor is loaded more than 5 seconds.</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_ERRREVPOWR</td>
<td>A counter electromotive force of the motor exceeds 70V</td>
<td>0x00002000</td>
</tr>
<tr>
<td>FFLAG_ERRMOTORPOWER</td>
<td>The motor is not connected to drive correctly.</td>
<td>0x00004000</td>
</tr>
<tr>
<td>FFLAG_ERRINPOSITION</td>
<td>After operation is finished, a position error occurs for more than 3 seconds.</td>
<td>0x00008000</td>
</tr>
<tr>
<td>FFLAG_EMGSTOP</td>
<td>The motor is under emergency stop.</td>
<td>0x00100000</td>
</tr>
<tr>
<td>FFLAG_SLOWSTOP</td>
<td>The motor is under general stop.</td>
<td>0x00200000</td>
</tr>
<tr>
<td>FFLAG_ORIGINRETURNING</td>
<td>The motor is returning to the origin.</td>
<td>0x00400000</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 operating to the normal speed, not acceleration / deceleration sections.</td>
<td>0x80000000</td>
</tr>
</tbody>
</table>
Next bit assignments can be applied over the Firmware version of 06.03.043.10

<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 error occurs.</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_SWPOGILMT</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>Reserved1</td>
<td></td>
<td>0X00000020</td>
</tr>
<tr>
<td>Reserved2</td>
<td></td>
<td>0X00000040</td>
</tr>
<tr>
<td>FFLAG_ERRPOSOVERFLOW</td>
<td>Position error is higher than ‘Pos Error Overflow Limit’ parameter after position command</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_ERRPOSTRACKING</td>
<td>Position error is higher than ‘Pos Tracking Limit’ parameter during position command run.</td>
<td>0X00000400</td>
</tr>
<tr>
<td>FFLAG_ERRPOSOVERLOAD</td>
<td>Load exceeding the max torque of the motor is loaded more than 5 seconds.</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_ERRBACKEMF</td>
<td>A counter electromotive force of the motor exceeds 70V.</td>
<td>0X00002000</td>
</tr>
<tr>
<td>FFLAG_ERRMOTORPOWER</td>
<td>The power supplied to the motor is less than low limited value.</td>
<td>0X00004000</td>
</tr>
<tr>
<td>FFLAG_ERRINPOSITION</td>
<td>After operation is finished, a position error occurs for more than 3 seconds.</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>FFLAG_INPOSITION</td>
<td>Imposition has been finished.</td>
<td>0X00080000</td>
</tr>
<tr>
<td>FFLAG_SERVOON</td>
<td>The motor is under Servo ON.</td>
<td>0X00100000</td>
</tr>
<tr>
<td>FFLAG_ALARMRESET</td>
<td>AlarmReset 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 motor operates to z-pulse type of origin return operations.</td>
<td>0X10000000</td>
</tr>
<tr>
<td>FFLAG_ORIGINRETOK</td>
<td>Origin return operation has been finished.</td>
<td>0X20000000</td>
</tr>
<tr>
<td>FFLAG_MOTIONDIR</td>
<td>To display the motor operating direction (+: Off, -: On)</td>
<td>0X40000000</td>
</tr>
<tr>
<td>FFLAG_MOTIONING</td>
<td>The motor is running.</td>
<td>0X80000000</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 operating to the normal speed, not acceleration / deceleration sections.</td>
<td>0X80000000</td>
</tr>
</tbody>
</table>
### 1-2-6. Position Table Item
Refer to ‘motion_define.h’ of include files.

<table>
<thead>
<tr>
<th>Name</th>
<th>Name of Structure Parameter</th>
<th>Number of Bytes</th>
<th>Offset</th>
<th>Unit</th>
<th>Low Limit</th>
<th>Upper Limit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>IPosition</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>5000000</td>
</tr>
<tr>
<td>High Speed</td>
<td>dwMoveSpd</td>
<td>4 (unsigned)</td>
<td>8</td>
<td>pps</td>
<td>0</td>
<td>5000000</td>
</tr>
<tr>
<td>Accl. 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>10</td>
</tr>
<tr>
<td>Wait time</td>
<td>wWaitTime</td>
<td>2 (unsigned)</td>
<td>18</td>
<td>msec</td>
<td>0</td>
<td>6000000</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>255</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
<td>10255</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>0</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>0</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>0</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>0</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>Check Inposition</td>
<td>bCheckInpos</td>
<td>2 (unsigned)</td>
<td>38</td>
<td></td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Compare Position</td>
<td>lTriggerPos</td>
<td>4 (signed)</td>
<td>40</td>
<td>pulse</td>
<td>-134217728</td>
<td>+134217728</td>
</tr>
<tr>
<td>Compare Width</td>
<td>wTriggerOnTime</td>
<td>2 (unsigned)</td>
<td>44</td>
<td>msec</td>
<td>1</td>
<td>9999</td>
</tr>
<tr>
<td>Push Ratio</td>
<td>wPushRatio</td>
<td>2 (unsigned)</td>
<td>46</td>
<td>%</td>
<td>10</td>
<td>100</td>
</tr>
<tr>
<td>Push Speed</td>
<td>dwPushSpeed</td>
<td>4 (unsigned)</td>
<td>48</td>
<td>pps</td>
<td>0</td>
<td>5000000</td>
</tr>
<tr>
<td>Push Position</td>
<td>lPushPosition</td>
<td>4 (signed)</td>
<td>52</td>
<td>pulse</td>
<td>-134217728</td>
<td>+134217728</td>
</tr>
<tr>
<td>Blank</td>
<td></td>
<td>8 (unsigned)</td>
<td>56</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」.

### 1-2-7. Information of Motors
Firstly the number and 2-3 characters are display the motor size and length.

**Example 1**  56XL : Motor Flange size is 56mm and Extra long size

Second area is display the motor maker information like 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>
</tbody>
</table>
1–3. Program Method

There are 2 method of programming for Ezi-SERVO Plus-R.

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

The second method can be accomplished by sending command characters directly to Ezi-SERVO Plus-R. The user have to prepare low level protocol programming like ‘Protocol Test’ program. This method is normally used for PLC system. For excise the protocol programming, ‘ProtocolTest_PlusR.exe’ GUI program is serviced together. Refer to 「3. Protocol for PLC Program」.
2. Library for PC Program(Ver6)

2-1. Library Configuration

To use this library, C++ header file(*.h) and library file(*.lib or *.dll) are required. These files are included in “afflech EziMOTION PlusR include”. The following contents should be included in a source file for development.

```cpp
#include “afflech EziMOTION PlusR include/FAS_EziMotionPlusR.h”
#include “afflech EziMOTION PlusR include/COMM_Define.h”
#include “afflech EziMOTION PlusR include/MOTION_DEFINE.h”
#include “afflech EziMOTION PlusR include/ReturnCodes_Define.h”
```

Also, library files are as follows:

“afflech EziMOTION PlusR include/EziMotionPlusR.lib”
“afflech EziMOTION PlusR include/EziMotionPlusR.dll”

A sample program source of using library is included in a “afflech EziMOTION PlusR Examples” folder.

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

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

<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_RUNFAIL</td>
<td>133</td>
<td>The motor is already running or not prepared for running.</td>
</tr>
<tr>
<td></td>
<td>FMP_RESETFAIL</td>
<td>134</td>
<td>The user cannot execute AlarmReset command while the servo is ON.</td>
</tr>
<tr>
<td></td>
<td>FMP_SERVOONFAIL1</td>
<td>135</td>
<td>An alarm has occurred.</td>
</tr>
<tr>
<td></td>
<td>FMP_SERVOONFAIL2</td>
<td>136</td>
<td>The motor is under Emergency Stop.</td>
</tr>
<tr>
<td></td>
<td>FMP_SERVOONFAIL3</td>
<td>137</td>
<td>ServoON'signal is already assigned to input pin.</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_PACKETCRCERROR</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 divide by 3 groups.

(1) Communication Error

- FMM_NOT_OPEN,
  COM Port is not connected.

- FMM_INVALID_PORT_NUM,
  COM Port number is not exist. Checking the ‘Device Manager’ window in Window OS.
FMM_INVALID_SLAVE_NUM,

Slave number is not exist. Checking the ID value of the drive.

FMC_DISCONNECTED = 5,

COM Port is disconnect 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_RECVPACKET_ERROR,
The length of received packet is not correct. Checking the possibility of noise on communication cable.

FMP_FRAMETYPEERROR = 0x80,
Drive do not recognize the command or wrong command is sended. Checking the command value that you want to send to the drive.

FMP_DATAERROR,
The value of the sended data is out of the proper range for 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 CRC value on drive is not correct. Checking the possibility of noise on communication cable.

(2) Wrong Command

FMP_RUNFAIL = 0x85,
Fail on motion command: The motor can not run on next status.
- The motor is already running
- The motor is under stop command
- Servo OFF status
- Try to Z-pulse Origin without external encoder (only for Ezi-STEP)

FMP_RESETFAIL,
Fail on reset command: The motor can not reset on next status.
- Servo ON status
- Already ‘Reset’ status by external input signal.
FMP_SERVOONFAIL1,
Wrong ‘Servo ON’ command during Alarm happens.

FMP_SERVOONFAIL2,
Wrong ‘Servo ON’ command during E-Stop happens.

FMP_SERVOONFAIL3,
‘Servo ON’ Signal is assigned by external input pin. In this case Servo ON command by DLL library is not working.

(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 return. Otherwise, FALSE will return.</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 return.</td>
</tr>
<tr>
<td>FAS_GetMotorInfo</td>
<td>The drive reads motor type and maker: Motor type and maker information will return.</td>
</tr>
<tr>
<td>FAS_IsSlaveExist</td>
<td>The drive checks whether there is the relevant drive: When it exists, TRUE will return. Otherwise, FALSE will return.</td>
</tr>
</tbody>
</table>
**FAS_Connect**

FAS_Connect is the function of connecting Ezi-SERVO Plus-R.

**Syntax**

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

**Parameters**

- **nPortNo**
  - Select a serial port to be connected.

- **dwBaud**
  - Input the Baudrate of the serial port.

**Return Value**

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

**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-SERVO 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 disconnect

**Remarks**

**Example**

Refer to 'FAS_Connect' library.

**See Also**

FAS_Connect
**FAS_GetSlaveInfo**

To get the version information string of the relevant drive

Syntax

```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`  
  Relevant drive type number
- `lpBuff`  
  Buffer pointer to get version information string
- `nBuffSize`  
  lpBuff memory allocation size

Return Value

- **FMM_OK**: Command has been normally performed.
- **FMM_NOT_OPEN**: 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`: Relevant motor type number
- `lpBuff`: Buffer pointer to get version information string
- `nBuffSize`: lpBuff memory allocation size

Return Value

- `FMM_OK`: Command has been normally performed.
- `FMM_NOT_OPEN`: 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**

To check that the drive is connected

**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
### 2-4. Parameter Control Function

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FAS_SaveAllParameters</td>
<td>Current parameters are saved to the ROM: 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>The designated parameter is saved to the RAM: Specific parameter is saved.</td>
</tr>
<tr>
<td>FAS_GetParameter</td>
<td>The designated parameter is read from the RAM: Specific parameter is read.</td>
</tr>
<tr>
<td>FAS_GetROMParameter</td>
<td>The designated parameter is read from the ROM: Specific parameter is read from the ROM.</td>
</tr>
</tbody>
</table>
FAS_SaveAllParameters

All parameters edited up to now & assign status of In/Out signals are saved in 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 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

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, SERVO_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-SERVO Plus-R.
        printf("Parameter [before] : Start Speed = %ld %d", lParamVal);
    }
}```
// Change Axis Start Speed parameter as 200 then read it again.
nRtn = FAS_SetParameter(nPortNo, iSlaveNo, SERVO_AXISSTARTSPEED, 200);
ASSERT(nRtn == FMM_OK);  // You have to check if the command didn't execute correctly.

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

// Check the value saved in the ROM.
nRtn = FAS_GetROMParameter(nPortNo, iSlaveNo, SERVO_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, SERVO_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 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` : There is no parameter of designated `iParamNo`.

Remarks

The function operates only for 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 values 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 imported
- **lParamValue**: Parameter values

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**: There is no parameter of designated iParamNo.

Remarks

The function operates only for 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 imported
- **lRomParam**
  - Parameter values saved in the ROM

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** : 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_ServoEnable</td>
<td>The Servo of the drive designated turns ON/OFF.</td>
</tr>
<tr>
<td>FAS_ServoAlarmReset</td>
<td>The drive which an alarm occurs is released:</td>
</tr>
<tr>
<td></td>
<td>Troubleshoot the alarm cause and use this function.</td>
</tr>
</tbody>
</table>
FAS_ServoEnable

To turn ON/OFF the drive servo

Syntax

```c
int FAS_ServoEnable(
   BYTE nPortNo,
   BYTE iSlaveNo,
   BOOL bOnOff
);
```

Parameters

- **nPortNo**: Port number of relevant drive
- **iSlaveNo**: Slave number of relevant
- **bOnOff**: Enable or Disable.

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

The given time is required until Servo ON flag in the axis status turns on after enable.

Example

```c
#include "FAS_EziMOTIONPlusR.h"

void funcAxisStatus()
{
   BYTE nPortNo = 1; // COMM Port Number
   BYTE iSlaveNo = 0; // Slave No (0 ~ 15)
   EZISERVO_AXISSTATUS AxisStatus:
   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;
   }

   nRtn = FAS_GetAxisStatus(nPortNo, iSlaveNo, &(AxisStatus.dwValue));
   _ASSERT(nRtn == FMM_OK);

   // If SERVO_ON flag turns off, the servo turns on.
   if (AxisStatus.FFLAG_SERVOON == 0)
   {
      nRtn = FAS_ServoEnable(nPortNo, iSlaveNo, TRUE);
      _ASSERT(nRtn == FMM_OK);
   }

   // If there is an alarm, AlarmReset runs.
   if (AxisStatus.FFLAG_ERRORALL || AxisStatus.FFLAG_ERROROVERCURRENT ||
      AxisStatus.FFLAG_ERROROVERLOAD)
   {
      nRtn = FAS_ServoAlarmReset(nPortNo, iSlaveNo);
      _ASSERT(nRtn == FMM_OK);
   }
}
// Disconnect.
FAS_Close(nPortNo);

See Also
FAS_ServoAlarmReset
**FAS_ServoAlarmReset**

To send AlarmReset command

**Syntax**

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

**Parameters**

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

**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**

Before sending this command, troubleshoot the alarm cause.
For alarm cause, refer to ‘User Manual_Text’.

**Example**

Refer to ‘FAS_ServoEnable’ library

**See Also**

FAS_ServoEnable
## 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>Input signal is set to [ON] or [OFF].</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 output port:</td>
</tr>
<tr>
<td></td>
<td>Output signal is set to [ON] or [OFF].</td>
</tr>
<tr>
<td>FAS_GetIoOutput</td>
<td>To read the current output 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 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 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 level:</td>
</tr>
<tr>
<td></td>
<td>Setting for each 9 variable signals is assigned to the Input and Output port.</td>
</tr>
<tr>
<td>FAS_IoAssignMapReadROM</td>
<td>To load the pin 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 ‘1-2. Structure of Frame Type’.

## 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 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

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 & SERVO_IN_BITMASK_LIMITP)
    {
        // Limit + input is ON.
    }
}
```
if (dwInput & SERVO_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, SERVO_IN_BITMASK_CLEARPOSITION | SERVO_IN_BITMASK_USERIN1, SERVO_IN_BITMASK_PJOG);
ASSERT(nRtn == FMM_OK);

// Check I/O output.
nRtn = FAS_GetIOOutput(nPortNo, iSlaveNo, &dwOutput);
ASSERT(nRtn == FMM_OK);
if (dwOutput & SERVO_OUT_BITMASK_USEROUT0)
{
    // User Output 0 is ON.
}

// Turn off User Output 1 and 2 signals.
nRtn = FAS_SetIOOutput(nPortNo, iSlaveNo, 0, SERVO_OUT_BITMASK_USEROUT1 | SERVO_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 ‘1-2. Structure of Frame Type’.

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 which input values will be saved

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

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

![Diagram](image)

When Port A is supplied 24V from an external input port, the input is recognized to 5V(High).

Example

Refer to ‘FAS_SetIOInput’ library.

See Also

FAS_SetIOInput
FAS_SetIOOutput

To read I/O output values. For more information, refer to ‘1-2. Structure of Frame Type’.

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
- `dwIOCLRMask` 
  Output bitmask value be cleared

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

There are 10 input pins in EziSERVO PlusR. The user can select and use 9 output pins of them.

![DIAGRAM](image)

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 ‘1-2. Structure of Frame Type’.

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 which the output value will be saved.

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

Refer to ‘FAS_SetIOInput’ library

See Also

FAS_SetIOOutput
FAS_GetIOAssignMap

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

Syntax

```c
int FAS_GetIOAssignMap(
    BYTE nPortNo,
    BYTE iSlaveNo,
    BYTE iiOPinNo,
    BYTE* nIOLogic,
    BYTE* bLevel
);
```

Parameters

- **nPortNo**
  Port number of relevant drive.
- **iSlaveNo**
  Slave number of relevant drive.
- **iiOPinNo**
  I/O pin number to be read
- **nIOLogic**
  Parameter pointer which the logic value assigned to a relevant pin will be saved
- **bLevel**
  Parameter pointer which the active level of relevant logic will be saved

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

For nIOLogic, 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 iPinNo;
    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 SERVOON 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, SERVO_IN_BITMASK_SERVOON,
    LEVEL_LOW_ACTIVE);
ASSERT(nRtn == FMM_OK);

// Check assign information of output pin.
for (i=0; i<10/*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,
    SERVO_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 ‘1-2. Structure of Frame Type’.

Syntax

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

Parameters

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

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**: Designated iIOPinNo or nIOLogic value is out of range.

Remarks

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

Example

Refer to ‘FAS_GSetIOAssignMap’ library

See Also

FAS_GetIOAssignMap
FAS_IOAssignMapReadROM

To load the status of CN1 assignment 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 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_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 actual command 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 to 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 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

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 he 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 to 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 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

The user sets the encoder feedback counter value to the value that he 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 that command position value will be saved

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

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-SERVO Plus-R.
    nRtn = FAS_GetCommandPos(nPortNo, iSlaveNo, &lValue);
    _ASSERT(nRtn == FMM_OK);
    printf("CMDPOS : %d\n", lValue);
    nRtn = FAS_GetActualPos(nPortNo, iSlaveNo, &lValue);
    _ASSERT(nRtn == FMM_OK);
    printf("ACTPOS : %d\n", lValue);
    nRtn = FAS_GetPosError(nPortNo, iSlaveNo, &lValue);
    _ASSERT(nRtn == FMM_OK);
    printf("POSERR : %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 which the actual position value will be saved.

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

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 which the position error value will be saved

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

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 which the actual velocity value will be saved

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

Refer to 'FAS_GetCommandPosition' library.

See Also
FAS_ClearPosition

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

Syntax

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

Parameters

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

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

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
<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 return.</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 return.</td>
</tr>
<tr>
<td>FAS_GetAllStatus</td>
<td>To read all status including 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_GetIOAxisStatus

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

Syntax

```c
int FAS_GetIOAxisStatus(
    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 which the I/O input value will be saved.
- **dwOutStatus**
  - Parameter pointer which the I/O output value will be saved.
- **dwAxisStatus**
  - Parameter pointer which the axis status value of the relevant motor will be saved

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

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 which the command position value will be saved.
- `lActPos` Parameter pointer which the actual position value will be saved.
- `lPosErr` Parameter pointer which the position error value will be saved.
- `lActVel` Parameter pointer which the actual velocity value will be saved.
- `wPosItemNo` Parameter pointer which current running item number in the Position Table will be saved.

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

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

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 which the I/O input value will be saved.
- **dwOutStatus**
  Parameter pointer which the I/O output value will be saved.
- **dwAxisStatus**
  Parameter pointer which the axix status value of the relevant motor will be saved.
- **lCmdPos**
  Parameter pointer which the command position value will be saved.
- **lActPos**
  Parameter pointer which the actual position value will be saved.
- **lPosErr**
  Parameter pointer which the position error value will be saved.
- **lActVel**
  Parameter pointer which the actual velocity value will be saved.
- **wPosItemNo**
  Parameter pointer which current running item number in the Position Table will be saved.

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

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 ‘1-2. Structure of Frame Type’.

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 which the axis status value of the relevant motor.

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

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>The motor in running is decelerate and stopped.</td>
</tr>
<tr>
<td>FAS_EmergencyStop</td>
<td>The motor in running stops directly without deceleration</td>
</tr>
<tr>
<td>FAS_MoveOriginSingleAxis</td>
<td>The motor starts the origin return.</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: This function is available to Jog motion.</td>
</tr>
<tr>
<td>FAS_PositionAbsOverride</td>
<td>While the motor is running, the target absolute position value [pulse] is changed.</td>
</tr>
<tr>
<td>FAS_PositionIncOverride</td>
<td>While the motor is running, the target incremental position value [pulse] is changed.</td>
</tr>
<tr>
<td>FAS_VelocityOverride</td>
<td>While the motor is running, the running velocity value [pulse] is changed.</td>
</tr>
<tr>
<td>FAS_AllMoveStop</td>
<td>All motors that connected in same port are decelerate and stopped.</td>
</tr>
<tr>
<td>FAS_AllEmergencyStop</td>
<td>All motors that connected in same port are directly stop without deceleration.</td>
</tr>
<tr>
<td>FAS_AllMoveOriginSingleAxis</td>
<td>All motors that connected in same port are starts the origin return.</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_MoveLinearAbsPos</td>
<td>More than 2 motors that connected in same port Linear Interpolation moves as much as the given absolute position value.</td>
</tr>
<tr>
<td>FAS_MoveLinearIncPos</td>
<td>More than 2 motors that connected in same port Linear Interpolation moves as much as the given incremental position value.</td>
</tr>
</tbody>
</table>
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 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

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 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

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 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

Refer to 'FAS_MoveSingleAxisAbsPos' library.

See Also
FAS_MoveSingleAxisAbsPos

To move the motor to the absolute coordinate

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 of position to move
- **lVelocity**
  - Velocity when the motor moves

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

```c
#include "FAS_EziMOTIONPlusR.h"

void funcMove()
{
    BYTE nPortNo = 1; // COMM Port Number
    BYTE iSlaveNo = 0; // Slave No (0 ~ 15)
    DWORD dwAxisStatus, dwInput;
    EZISERVO_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 and Servo ON status.
    nRtn = FAS_GetAxisStatus(nPortNo, iSlaveNo, &dwAxisStatus);
    _ASSERT(nRtn == FMM_OK);
    stAxisStatus.dwValue = dwAxisStatus;

    //if (dwAxisStatus & 0x00000001)
    if (stAxisStatus.FFLAG_ERRORALL)
```
Check input status.
nRtn = FAS_GetIOInput(nPortNo, iSlaveNo, &dwInput);
    _ASSERT(nRtn == FMM_OK);

if (dwInput & (SERVO_IN_LOGIC_STOP | SERVO_IN_LOGIC_PAUSE | SERVO_IN_LOGIC_ESTOP))
    FAS_SetIOInput(nPortNo, iSlaveNo, 0, SERVO_IN_LOGIC_STOP | SERVO_IN_LOGIC_PAUSE | SERVO_IN_LOGIC_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 of position to move
- **lVelocity**
  - Velocity when the motor moves

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

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 which the motor moves (0: -Limit, 1: +Limit)

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

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
FAS_MoveVelocity

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

Syntax

```
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 which the motor moves (0: -Jog, 1: +Jog)

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

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 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

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

![Graph showing Position Override and Speed over Time](image1)

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

![Graph showing Position Override and Speed over Time](image2)
3) If the changed target position is set to the closer coordinate than the original target position, the motor once stops to the position before change and then performs acceleration and deceleration to stop 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 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 ‘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 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

1) In case of ((change speed) < (speed before change)), the motor reaches the change speed through acceleration/deceleration using a new velocity pattern.
5) In case of ((change speed) ≥ (speed before change)), the motor reaches the change speed through acceleration/deceleration without any new velocity pattern.
4) The motor reaches the ‘speed before change’ without a change of the velocity pattern and then it reaches the ‘change speed’ by a new velocity pattern.
2),3) 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 the motor that connected in same port.

Syntax

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

Parameters

- `nPortNo`: Port number of relevant drive.
- `iSlaveNo`: Slave number of relevant drive. (must be ‘99’)

Return Value

No response

Remarks

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
**FAS_AllEmergencyStop**

To stop the motor that connected in same port without deceleration

**Syntax**

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

**Parameters**

`nPortNo`
- Port number of relevant drive.

`iSlaveNo`
- Slave number of relevant drive. (must be ‘99’)

**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 that is connected in same port. For more information, refer to 'User Manual_Text 9.3 Origin Return'.

Syntax

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

Parameters

- `nPortNo`
  - Port number of relevant drive.
- `iSlaveNo`
  - Slave number of relevant drive. (must be ‘99’)

Return Value

- No response

Remarks

Example

Refer to ‘FAS_MoveSingleAxisAbsPos’ library.

See Also
FAS_AllMoveSingleAxisAbsPos

To move the motor that connected in same port to the absolute coordinate

Syntax

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

Parameters

- `nPortNo`
  Port number of relevant drive.
- `iSlaveNo`
  Slave number of relevant drive. (must be ‘99’)
- `lAbsPos`
  Absolute coordinate of position to move
- `lVelocity`
  Velocity when the motor moves

Return Value

- No response

Remarks

Example

See Also
FAS_AllMoveSingleAxisIncPos

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

Syntax

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

Parameters

- `nPortNo` Port number of relevant drive.
- `iSlaveNo` Slave number of relevant drive. (must be '99')
- `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_MoveLinearAbsPos / FAS_MoveLinearIncPos

To move (Linear Interpolation) more than 2 motors that connected in same port to the absolute / incremental coordinate.

Syntax

```c
int FAS_MoveLinearAbsPos(
    BYTE nPortNo,
    BYTE nNoOfSlaves,
    BYTE *iSlaveNo,
    long *lAbsPos,
    DWORD lFeedrate,
    WORD wAccelTime
);
```

Parameters

- **nPortNo**
  - Port number of relevant drive.
- **nNoOfSlaves**
  - Slave numbers for Linear motioning.
- **iSlaveNo**
  - Array of Slave numbers.
- **lAbsPos**
  - Array of position value for each slave. (in case of incremental : Distance)
- **lFeedrate**
  - Speed value for motioning.
- **wAccelTime**
  - Acceleration & deceleration time value.

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

When this function is used, the ‘Start speed’ value must be designated to ‘1’.
## 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 all items of position table</td>
</tr>
<tr>
<td>FAS_PosTableWriteItem</td>
<td>To save specific all items of position table items 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 one item of position table items 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 which item value is saved

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

```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 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.
- `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 acts 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 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.
- `FMC_POSTABLE_ERROR` : An error occurs while position table is being saved.

Remarks

Position table data is saved to RAM / ROM area. This function acts 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 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.
- **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 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_GetAllStatus
- FAS_MoveStop
- FAS_EmergencyStop
**FAS_PosTableReadOneItem**

To read a one 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 in PT items. (Refer to ‘1-2-6. Position Table Item’)

**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_PosTableWriteOneItem
FAS_PosTableWriteOneItem

To edit one item in the specific position table

Syntax

```c
int FAS_PosIableWriteOneItem(
    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 save in PT items. (Refer to ‘1-2-6. Position Table Item’)

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.
- **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>
<tr>
<td>FAS_MovePush</td>
<td>To request push motion(maintain specified motor torque) command</td>
</tr>
<tr>
<td>FAS_GetPushStatus</td>
<td>To request the current push motion status</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]
- **DWORD dwPeriod**
  - Period of output signal [pulse]
- **DWORD 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
FAS_MovePush

To request push motion (maintain specified motor torque) command

Syntax

```c
int FAS_MovePush(
    BYTE nPortNo,
    BYTE iSlaveNo,
    DWORD dwStartSpd,
    DWORD dwMoveSpd,
    long lPosition,
    WORD wAccel, WORD wDecel,
    WORD wPushRate,
    DWORD dwPushSpd,
    long lEndPosition
);
```

Parameters

- `nPortNo` Port number of relevant drive.
- `iSlaveNo` Slave number of relevant drive.
- `dwStartSpd` Start speed of position command.
- `dwMoveSpd` Move speed of position command.
- `lPosition` Absolute target position of position command.
- `wAccel` Accel time of position command.
- `wDecel` Deceleration time of position command.
- `wPushRate` Torque ratio of Push motion.
- `dwPushSpd` Move speed of Push motion.
- `lEndPosition` Absolute target position of push motion.

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_GetPushStatus
FAS_GetPushStatus

To request the current push motion status

Syntax

```c
int FAS_MovePush(
    BYTE nPortNo,
    BYTE iSlaveNo,
    BYTE* nPushStatus
);
```

Parameters

- `nPortNo` Port number of relevant drive.
- `iSlaveNo` Slave number of relevant drive.
- `BYTE* nPushStatus` Status value of push motion. (refer to '1-2-1. FrameType and Data Configuration')

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_Move Push
3. Protocol for PLC Program

Next window is open when you click icon in User Program(GUI) installed folder.

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

(1) Servo ON/OFF command

1) Insert 'Comm Port' number and click 'Connect' button.
2) Header: Click 'Header' and you can see '[0xAA][0xCC]' on 'Send Buffer' window.
3) Slave ID: Insert your slave number (above example is '0') and click 'SlaveNo'.
4) Frame type: Insert 'Frame type'.
   You can find next table information in '1-2-1. Frame Type and Data Configuration'
   on UserManual(EziSERVO PlusR)_Communication Function.
5) Data: Insert '1' in area and click '1 byte'.
6) CRC: Click 'CRC' and the 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 informations from Ezi-SERVO
   Plus-R on 'Buffer Received' window.

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

The header and tail information is needed for protocol programming.
Additionally Frame Data (Slave ID, Frame type, Data and CRC) is also needed in every protocol with header and tail.
(2) Motion 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: After sending command you can check the motor rotation and if click ‘Send’ more the motor will rotate one more time.

(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.
