

Animatics Corporation

Class 5 Motor with Profibus Interface

Specifications

Rev. F

Rev D: CV -

- 1) Correction to RI(0), read object 128
- 2) Correction to index arming commands.

Rev E: CV -

- 1) Correction to hex values in command code list Page 23 and Page 24.
- 2) Description of CANCTL(11,x) for Class 4 emulation.
- 3) Other relevant Class 4 emulation detail added to command and response objects.
- 4) Status bit section on page 9 to describe Class 4 emulation bits.

Ref F: CV - May 4, 2011

Documents changes added with 5.32.3.14 version firmware.

- 1) Added Profibus read access to user status bits using status words 12 and 13. (Read objects 182 and 183)
- 2) Added Profibus read access to status word 16 for I/O. (Read object 186)
- 3) Added Profibus set/clear of user status bits. (Write objects 90-95)

Contents:

3	Overview
3	Equipment Required
4	ProfiBus Connector Pinouts
5	Connect to ProfiBus Example
7	<i>Configure Motor with PC</i>
7	<i>User Program Required in Motor</i>
7	<i>Non-Volatile EEPROM Values Required in Motor</i>
7	<i>Configure PLC with PC</i>
9	<i>Status Bits</i>
10	Sample ProfiBus Command Sequences
16	Non-Volatile Data
16	<i>Motor Non-Volatile Data</i>
17	ProfiBus Packets
17	<i>ProfiBus Output and Input Packet Format</i>
18	<i>Command (Output) Packet Notes</i>
19	<i>Response (Input) Packet Notes</i>
20	Alternate Serial Channel
21	Appendix:
21	ProfiBus Packet Command and Response Codes
21	<i>Command Packet Command Codes to Motor Commands</i>
28	<i>Response Packet Codes to Motor Commands</i>
33	Troubleshooting
34	<i>ProfiBus LEDs</i>

Overview

ProfiBus is an independent, open fieldbus standard that allows different manufacturers of automation products to communicate without special interface adjustments. Specifically, ProfiBus-DP, which is optimized for high speed is designed to communicate between control systems and distributed I/O at the device level.

Animatics has defined a set of 8-bit command and response codes to be transmitted and received over ProfiBus. These codes generally correspond to Class 5 SmartMotor™ commands. To set target position, for example, the command code to set target position is transmitted together with the data consisting of the target position value.

The ProfiBus SmartMotor™ is a SmartMotor™ with the addition of a ProfiBus connector which accepts commands as a slave over a ProfiBus DP network. In addition to communicating over ProfiBus, commands in the Class 5 SmartMotor™ ASCII “Ani-basic” language may be sent through the channel 0 RS232 port, or channel 1 RS485 port.

Document sections include Output and Input data formats (ProfiBus cargo), a list of Animatics ProfiBus command codes explained in terms of the equivalent SmartMotor™ ASCII “Ani-Basic” commands, and a list of Animatics ProfiBus response codes, explained in terms of the equivalent SmartMotor™ ASCII “Ani-Basic” commands.

Handshaking of the ProfiBus communication allows the output data to be coherently assembled by a PLC while the ProfiBus Master is continually transmitting the record buffer contained in PLC memory to the DP slave (motor).

Certain configuration data is held in non-volatile storage in the motor or in the motor’s ProfiBus gateway. The motor data eeprom must be correctly initialized prior to ProfiBus operation.

The ProfiBus GSD configuration file “DEAF070C.GSD” is necessary for the host to configure the ProfiBus Master to connect to the slave motor.

Equipment Required

You will need the following equipment to proceed:

HARDWARE

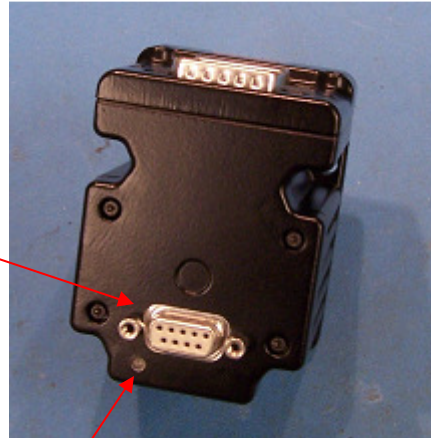
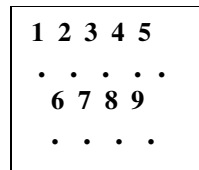
- Animatics ProfiBus SmartMotor™
- Animatics power supply or equivalent
- Animatics power/serial communications cable or equivalent
- PLC with ProfiBus master, or other PROFIBUS master
- “Official” ProfiBus cable
- “Official” ProfiBus connectors with correct terminating resistors
- PC with Windows 95 or later operating system and an RS232 serial port, or equivalent
- Animatics RS232 to RS485 converter if a Version 5 SmartMotor™ is used

SOFTWARE

- PLC configuration software
- Animatics SMI software
- Animatics ProfiBus GSD file “DEAF070C.GSD”

ProfiBus Connector Pinouts

DSUB face view:



- 1 nc
- 2 nc
- 3 BUS-B red
- 4 nc
- 5 nc
- 6 nc
- 7 nc
- 8 BUS-A green
- 9 nc

ProfiBus
Connection
Status LED

(With Termination and bias)

- 1 nc
- 2 nc
- 3 BUS-B red
- 4 nc
- 5 ground
- 6 +5v
- 7 nc
- 8 BUS-A green
- 9 nc

- 3-6 bias BUS-B to +5v
- 8-5 bias BUS-A to ground
- 3-8 terminator resistor between BUS-A and BUS-B

(Values measured in Siemen's connectors are:

Siemens bias resistors: 500 ohms
Siemens terminator resistor: 220 ohms)

Connect to ProfiBus Example

The example illustrates communication over ProfiBus by sending commands from a PLC over ProfiBus to cause the motor to continually report its changing clock value to the PLC over ProfiBus. The value displayed by the PLC registers containing the Profibus data received from the motor changes as the updated clock value is received from the motor.

Note: your motor type may vary from motor pictured below.

The steps, explained in more detail below, are:

1. Configure the motor (figure 1) through its serial port using a PC or laptop that is running Animatics SMI software to:
 - a. Set Profibus node number in motor non-volatile storage
2. Configure your PLC (figure 2) through its serial port using a PC or laptop that is running your PLC configuration software to:
 - a. Load the motor's Profibus GSD file
 - b. Assign and display the PLC registers associated with the motor's Profibus input and output data
3. Connect Profibus cable to the PLC and the motor.
4. Power cycle the motor to initialize the motor with the configured values.
5. Enter Profibus motor command to report motor clock in the PLC Profibus data registers using a PC or laptop that is running your PLC software with your PLC online (figure 3).
6. Witness clock value being updated in your PLC Profibus input registers, using a PC or laptop that is running your PLC software with your PLC online.

Please refer to other sections of the manual for details on sending command sequences and communication handshaking.

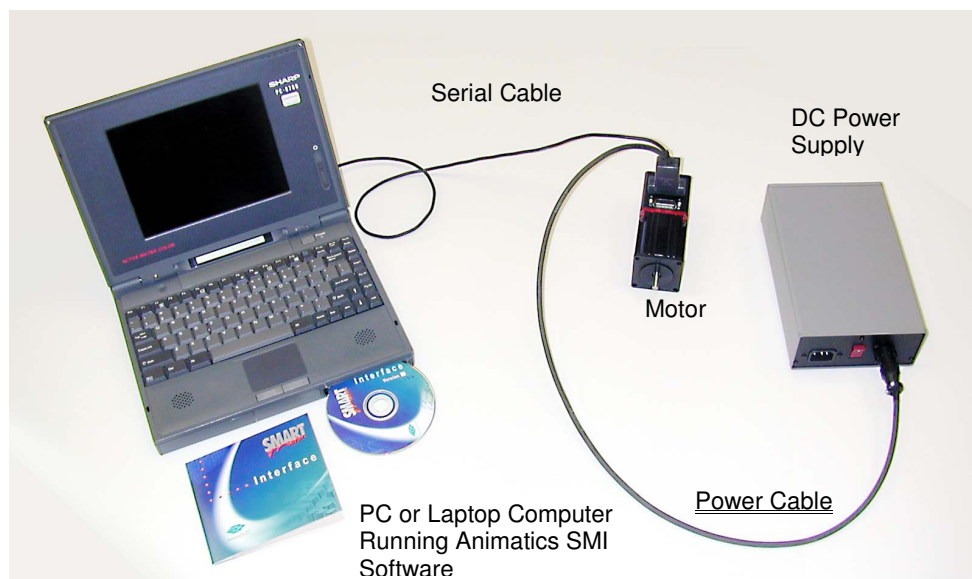


Figure 1 Configuring the Motor



Figure 2 Configuring the PLC

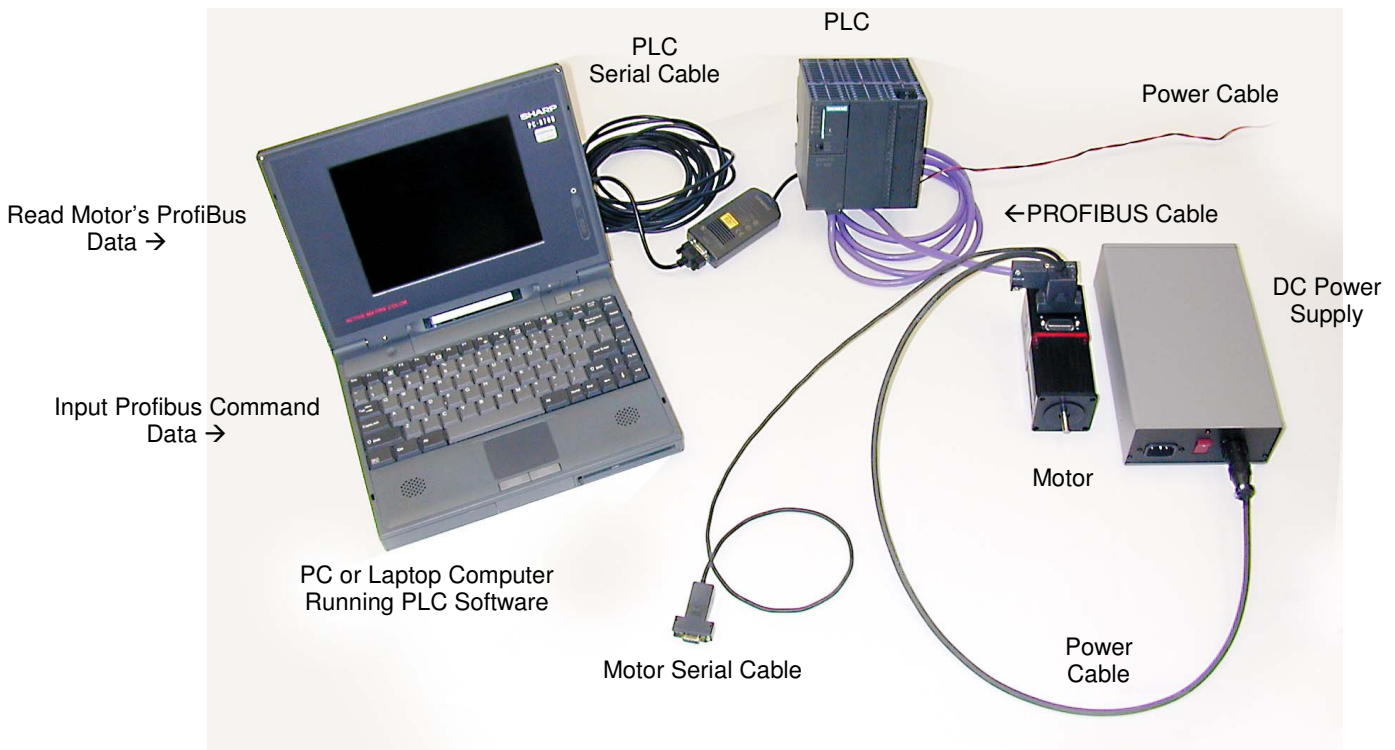


Figure 3 Communicating Over Profibus

Configure Motor with PC

Connect the motor to power supply (figure 1).

You may skip the balance of this step if the motor is already configured.

Connect the motor to the PC (figure 1).

Launch the Animatics SmartMotor™ Interface software (SMI, or SMI 2 for Version 5 motors).

User Program Required in Motor

No user program is specifically required by the Class 5 Profibus motor.

Non-Volatile EEPROM Values Required in Motor

If not already set, issue the following commands to set the following values:

```
Set the Profibus node ID to 3:  
CADDR=3
```

```
Set the polling rate as fast as possible. The first number is the command to set poll  
rate, the second number is the number of microseconds to wait between polling loops.  
CANCTL(8,0)
```

```
Set the action to take on network lost.  
CANCTL(9,0)
```

```
Set the program label to jump to on network lost (if selected):  
CANCTL(6,0)
```

Reserved Motor Variables:

The Class 5 Profibus motor does not need to reserve any user variables as the Class 4 did. The variables yyy and zzz are not affected unless deliberately accessed by the user. EPTR may be used by the user and is not modified by Profibus activity as the Class 4 was. VST and VLD commands are also independent Profibus.

Configure PLC with PC

You may skip this step if the PLC is already configured.

Using the PLC configuration software running in a PC, load the motor's GSD file, DEAF070C.GSD, set it up as a Profibus node, and define the node number of the motor (figure 2).

Set up the PLC memory that is the three words (six bytes) Profibus output to the motor and the seven words (fourteen bytes) input from the motor.

Connect Motor to ProfiBus

If the Profibus plugs have terminator switches, make sure the terminator switches at both ends of the Profibus bus are turned on, and all other terminator switches are off. It is NECESSARY for Profibus to have the proper termination and bias resistors. Plug the Profibus connector into the motor (figure 3).

Power-up, or power cycle the motor. For 2 seconds the ProfiBus LEDs on the motor in the proximity of the ProfiBus connector will have meaningless states. If the PLC is running, within a few seconds the ProfiBus connection status LED on the motor will turn GREEN, indicating the ProfiBus Master has established a connection with the slave motor.

PLC Sends Commands To Motor

Program the PLC, or modify by hand the PLC memory areas as described below to send the desired commands over ProfiBus to the gateway to communicate with the motor. Below are sequences of commands sent, showing all the intermediary ProfiBus packet output data states. Bold characters indicate changes in PLC memory output buffer and input buffer values.

Data Format

Each byte below is represented as two hexadecimal characters. For example, 7A represents hex 7A, or decimal 122.

Command Code	Response code	Data		Command Code Acknowledgement	Response Code Acknowledgement	Response Data	Status Word	Measured Position	Position Error
00	7A	0000 0000		00	00	0000 0000	0086	0000 0000	0000

Status bits when in Class 5 mode (default):

Bit: Description:

0	Busy Trajectory
1	Historical + Limit (hardware and software limit)
2	Historical - Limit (hardware and software limit)
3	Index report available for the rising edge of internal encoder
4	Position wrap around occurred
5	Position error fault
6	Temperature limit fault
7	Drive Off
8	Index input active
9	+ limit active (hardware and software limit)
10	- limit active (hardware and software limit)
11	Communication error of any type
12	Reserved
13	Command error (includes math and array errors)
14	Peak over-current occurred
15	Drive ready

Status bits when in Class 4 emulation mode:

Bit: Description:

0	Busy Trajectory
1	Historical + Limit (hardware and software limit)
2	Historical - Limit (hardware and software limit)
3	Index report available for the rising edge of internal encoder
4	Position wrap around occurred
5	Position error fault
6	Temperature limit fault
7	Drive Off
8	Index input active
9	+ limit active (hardware and software limit)
10	- limit active (hardware and software limit)
11	Math overflow
12	Array index error
13	Syntax error
14	Peak over-current occurred
15	Program checksum error

PLC Memory:

Each byte below is represented as two hexadecimal characters. For example, 86 represents hex 86, or decimal 134.

Output to slave motor
3 two-byte words out

Input from slave motor
7 two-byte words in

0000 0000 0000 0000 0000 0000 0086 0000 0000 0000

A status word of 0x0086 indicates servo off, left and right limits have been activated, drive not ready.

SEQUENCE TO SET REPORT DATA TO MOTOR CLOCK

Commands:

PROFIBUS COMMAND CODE	RESPONSE CODE	DATA	MOTOR COMMAND
	0x7A		RCLK

Insert response code 0x7A in output buffer (which is being transmitted continuously (ie cyclically) by the master to the slave motor. See Appendix: ProfiBus Packet Command and Response Codes to find response code RCLK and its value, hex 7A.

007A 0000 0000 0000 0000 0000 0086 0000 0000 0000

Wait for response code acknowledge in the input buffer (which is being received continuously (ie cyclically) by the master as a response from the slave motor. Clock data begins being cyclically updated.

007A 0000 0000 007A 0000 03A1 0086 0000 0000 0000

As time goes on, clock data is updated.

007A 0000 0000 007A 0001 B01A 0086 0000 0000 0000

Sample ProfiBus Command Sequences

These sequences serve to illustrate:
 disabling limits from preventing motion;
 turning the shaft in torque mode;
 moving a relative distance;
 command and response codes; and
 handshaking of messages.

Command and Response Codes

The command and response codes may be found in the appendix “ProfiBus Packet Command and Response Codes.” The symbolic command and response codes are listed, together with their values and the related SmartMotor™ “Ani-Basic” command. See the section “ProfiBus Packets” for further explanation of how to use the command and response codes.

Handshaking of Messages

Handshaking of output message changes is included in the protocol to ensure coherence in the packet. See the section “ProfiBus Packets” for an explanation of handshaking.

Disabling Limits from Preventing Motion

At power up, if limit switches are not connected to the motor, the electrical state of the limit pins will default to indicate that the motor is at the limits. This will prevent motion unless the limits are disabled, and a limit fault that might have occurred prior to disabling the limits is cleared.

These commands may be included in the user program downloaded to the motor that runs at power up. If a user program does NOT include commands to disable limits and clear fault status (or limits are NOT held inactive at power-up), perform the appendix command sequence titled:

SEQUENCE TO DISABLE LIMITS AND CLEAR FAULT STATUS IN CLASS 5 MOTOR

prior to attempting to cause the shaft to turn.

Turning the Shaft

If the above-referenced command sequence has been performed, or if the motor user program includes commands to disable limits and clear fault status (or limits are held inactive at power-up), the shaft may be made to turn by the following command sequences titled:

SEQUENCE TO INITIATE MODE TORQUE IN CLASS 5 MOTOR

SEQUENCE TO INITIATE RELATIVE POSITION MOVE IN CLASS 5 MOTOR

SEQUENCE TO DISABLE LIMITS AND CLEAR FAULT STATUS IN CLASS 5 MOTOR

Commands:

COMMAND CODE	RESPONSE CODE	DATA	RESULTING MOTOR COMMAND
0x01		0x30	EIGN(2)
0x01		0x33	EIGN(3)
0x01		0x44	ZS

PLC Memory:

Output to slave motor
3 words out

Input from slave motor
7 words in

0000 0000 0000 0000 0000 0000 0086 0000 0000 0000

Command Code	Response code	Data	Command Code Acknowledgement	Response Code Acknowledgement	Response Data	Status Word	Measured Position	Position Error
00	7A	0000 0000	00	00	0000 0000	0086	0000 0000	0000

Disable positive limit, command EIGN(2)

Insert command command EIGN(2) data = 0x30 in output buffer (which is being transmitted continuously (ie cyclically) by the master to the slave motor.

0000 0000 0030 0000 0000 0000 0086 0000 0000 0000

Set command code 0x01 in the output buffer.

0100 0000 0030 0000 0000 0000 0086 0000 0000 0000

Wait for command code acknowledge in the input buffer (which is being received continuously [ie cyclically]) by the master as a response from the slave motor.

0100 0000 0030 **0100** 0000 0000 0086 0000 0000 0000

The command code acknowledge is confirmation the gateway has received the command to transmit to the motor.

Clear command code in output buffer to handshake the next command.

0000 0000 0030 0100 0000 0000 0086 0000 0000 0000

Wait for the acknowledge of the cleared command code.

0000 0000 0030 **0000** 0000 0000 0086 0000 0000 0000

SEQUENCE TO INITIATE MODE TORQUE IN CLASS 5 MOTOR

Commands:

COMMAND CODE	RESPONSE CODE	DATA	RESULTING MOTOR COMMAND
0x94	0xA2	3072 (0x0c00)	T=3072 RVA (polled motor response)
0x01	0xA2	0x21	MT RVA (polled motor response)

PLC Memory:

**Output to slave motor
3 words out**

**Input from slave motor
7 words in**

0000 0000 0000 0000 0000 0000 0080 0000 0000 0000

Set torque value, command T=3072, specify response data to be current velocity

Begin to set torque T=3072 by putting **x 00 00 0C 00** in output data:

0000 **0000 0C00** 0000 0000 0000 0080 0000 0000 0000

Insert command code **0x94** and response code **0xA2**.

94A2 0000 0C00 0000 0000 0000 0080 0000 0000 0000

Wait for acknowledge in input buffer.

94A2 0000 0C00 **94A2** 0000 0000 0080 0000 0000 0000

Now T=3072 (0x0c00) and the response data value will be velocity. Clear the command code to handshake for the next command.

00A2 0000 0C00 94A2 0000 0000 0080 0000 0000 0000

Wait for acknowledge of command code clear in input buffer:

00A2 0000 0C00 **00A2** 0000 0000 0080 0000 0000 0000

Initiate torque mode, command MT

Now insert command 0x21 data to begin torque mode:

00A2 **0000 0021** 00A2 0000 0000 0080 0000 0000 0000

Now insert command code = **0x01**

01A2 0000 0021 00A2 0000 0000 0080 0000 0000 0000

Motor shaft will begin turning, if motor is not in a fault state, when command is received by gateway and passed on to motor.

Wait for command code acknowledge in the input buffer. Velocity becomes non-zero, reported as **0x00 14 00 00** in this example. Status changes, reported as **0x0009** in this example. Position becomes non-zero, reported as **0x00 00 00 A2** in this example.

Non-Volatile Data

Motor Non-Volatile Data

The motor stores some information about the Profibus network in eeprom, and must be initialized with the following data. This is accomplished through serial channel 0, using the CANCTL(function, value), and CADDR= commands. It would be possible to modify these values through ProfiBus, once a connection to the ProfiBus Master had been established. These commands are described in the “Profibus Packet Commands” section of this document.

The Class 5 Profibus interface is not a gateway expansion in the same way the Class 4 Profibus expansion was. Therefore, these values are integrated more into the Class 5 motor with unique commands. These

These commands write the following information to the motor’s non-volatile memory to be read at power-up.

Command	description/ parameter	Values
CADDR=	Profibus node (macID)	typically 3 to 127 63 is the factory default.
CANCTL(1,0)	Reset Profibus	Resets the Profibus hardware and protocol stack.
CANCTL(6, <value>)	NET_LOST_LABEL	Program label to jump to if the NET_LOST_LABEL option is chosen from the NET_LOST_ACTION function. This function has no effect if the NET_LOST_ACTION is anything other than NET_LOST_LABEL.
CANCTL(7,<value>)	Axis identifier (not mac ID)	A 32-bit value settable and readable via Profibus. Not required for normal operation. May be ignored.
CANCTL(8, <value>)	POLL_RATE	Polling rate of the Profibus service in microseconds. 0 is as fast as possible. Note that full microsecond resolution is not available, but the service time is typically under 1 millisecond. This setting allows for slowing down Profibus service to allow more CPU time for user programs.
CANCTL(9,<value>)	NET_LOST_ACTION	Action to take if Profibus network is lost (Master is no longer communicating to slave, timeout of master occurs in the gateway) 0 - IGNORE 1 - send command OFF to motor 2 - send command X to motor (soft stop) 3 - send command S to motor (immediate stop) 4 - send command GOSUB(x), where x is the value of NET_LOST_LABEL 5 - send command GOTO(x), where x is the value of NET_LOST_LABEL
CANCTL(10,<value>)	SET_PA_FIELD	Configure the Profibus input packet to use an alternate data source for the ‘Measured position’ field (words 4,5). <value>: 0 - report actual position in encoder counts. This is the power-up default value. 1 - report al[0] (big-end format.) 2 - report af[0] (IEEE-754 32-bit single precision. Big-end format.)
CANCTL(11,<value>)	Class 4 emulation	Configure the motor to perform certain actions similar to Class 4. Value is reset to 0 (Class 5 mode) when motor is reset. This value is not stored in the EE. 0 - Behave like a Class 5. (power-on default.) 1 - Perform certain actions like Class 4: Status word contains math error instead of communication error and array error, these are cleared by ZS. Status word index capture bit resets on index read, index re-armed by the read. Status word checksum error instead of drive ready. CMD_Zd works through Profibus to clear math overflow. CMD_Zu works through Profibus to clear array error.

ProfiBus Packets

ProfiBus Output and Input Packet Format

Output Data Format

Word	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	Command Code							
	1	Response Code							
1	2	Command Data Value (32 bits), Big Endian format							
	3								
2	4								
	5								

Input Data Format

Word	Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	Command Code Acknowledge							
	1	Response Code Acknowledge							
1	2	Response Data Value (32 bits), Big Endian format							
	3								
2	4								
	5								
3	6	Status Word (16 bits), Big Endian format							
	7								
4	8	Measured Position (32 bits), Big Endian format. Note: this field can be configured to report al[0], or af[0].							
	9								
5	10								
	11								
6	12	Position Error (16 bits), Big Endian format							
	13								

Command Code

Indicates a command to be issued to the SmartMotor. Also see Command Data Value.

Response Code

Indicates additional data to be included in the Response Data Value of the Input Data.

Command Data Value

Indicates the 32-bit value to be used in conjunction with the Command Code.

Command Code Acknowledge

Returned in the Input Data to indicate that a Command Code has been processed.

Response Code Acknowledge

Returned in the Input Data to indicate that a Response Code has been processed and that the current Response Data Value corresponds to that Response Code.

Response Data Value

32-bit value returned in the Input Data in response to a Response Code.

Status Word

SmartMotor's current status word (16-bit) (result of RW or RPW command).

Measured Position

SmartMotor's current measured position value (32-bit) (result of RP or RPW command).

Position Error

SmartMotor's current commanded trajectory position less current measured position.

Command (Output) Packet Notes

- Note 1: A command is issued to the SmartMotor exactly one time after the Command Code or Command Data Value changes in the output data. The proper sequence to issue a command is:
- set the Command Code to 0
 - wait for Command Code Acknowledge = 0
 - set the Command Data Value to the desired value
 - set the Command Code to the desired command
 - wait for Command Code Acknowledge = Command Code
- Note 2: Wherever you see <value>, insert the Command Data Value.
- Note 3: Wherever you see <a to z>, insert the lowercase letter corresponding to the value of u8VarIndexSet.
- Note 4: Wherever you see <index>, insert the array index stored in u8ArrIndexSetActual.
- Note 5: Wherever you see <length>, insert the length stored in u8VarLenSet or u8ArrLenSet.
- Note 6: Curly brackets {} indicate binary data rather than ASCII characters.
- Note 7: The Polling Rate (u32PollRate) is the minimum time (in microseconds) to wait between sending status requests, position requests, and requests associated with the Response Codes and commands. The microsecond scaling of this value is to support the higher-speed nature of the Class 5 Profibus interface. It does not imply that there is a response time with a resolution of single microseconds. Setting this value to 0 polls the Profibus interface as fast as possible, which is approximately 300-500 microseconds.
- Note 8: The SmartMotor variable yyy is no longer used by the Profibus module in the Class 5 motor, as compared to the Class 4 Profibus interface. The Class 5 memory map also differs that ab[200], aw[100] and al[50] do not share space with variable yyy. The means that those array locations are also not affected by the Profibus interface.
- Note 9: The Class 5 ProfiBus interface *does not* use the SmartMotor's EPTR during initialization to read startup parameters from the SmartMotor. The user program may use EPTR at the same time. Therefore, zzz is not used by the Class 5 Profibus interface, unlike the Class 4 Profibus gateway.
- Note 10: If an invalid value of the MAC ID (not 0-125) is found at startup, then the ProfiBus Default Address of 126 will be used.

Response (Input) Packet Notes

- Note 1: The requests associated with any Response Codes other than 214-225 are issued to the SmartMotor continuously (or according to the polling rate if set). When the Response Code in the output data transitions to a value in the range of 214-225, the associated request will be issued to the SmartMotor exactly one time after transition to one of these values. The proper sequence to issue a request for data is:
- set the Response Code to 0
 - wait for Response Code Acknowledge = 0
 - set the Response Code to the desired value
 - wait for Response Code Acknowledge = Response Code read data from Response Data Value (repeat as desired if not Response Codes 214-225)
- Note 2: The Polling Rate (u32PollRate) is the minimum time (in microseconds) to wait between sending status requests, position requests, and requests associated with the Response Codes and commands. The microsecond scaling of this value is to support the higher-speed nature of the Class 5 Profibus interface. It does not imply that there is a response time with a resolution of single microseconds. Setting this value to 0 polls the Profibus interface as fast as possible, which is approximately 300-500 microseconds.
- Note 3: Wherever you see <value>, insert the Response Data Value.
- Note 4: Wherever you see <a to z>, insert the lowercase letter corresponding to the value of u8VarIndexGet.
- Note 5: Wherever you see <index>, insert the array index stored in u8ArrIndexGetActual.
- Note 6: Wherever you see <length>, insert the length stored in u8VarLenGet or u8ArrIndexGet.
- Note 7: Curly brackets {} indicate binary data rather than ASCII characters.
- Note 8: The Response Data Value for a GET_MODE (SmartMotor RMODE) command will contain the integer code returned by the SmartMotor, unlike the Class 4 Profibus interface.
- Note 9: The SmartMotor variable yyy is no longer used by the Profibus module in the Class 5 motor, as compared to the Class 4 Profibus interface. Therefore, the user program *may* use variable yyy. The Class 5 memory map also differs that ab[200], aw[100] and al[50] do not share space with variable yyy. This means that those array locations are also not affected by the Profibus interface.
- Note 10: The Class 5 ProfiBus interface *does not* use the SmartMotor's EPTR during initialization to read startup parameters from the SmartMotor. The user program may use EPTR at the same time. Therefore, zzz is not used by the Class 5 Profibus interface, unlike the Class 4 Profibus gateway.
- Note 11: If an invalid value of the MAC ID (not 0-125) is found at startup, then the ProfiBus Default Address of 126 will be used.

ALTERNATE SERIAL CHANNEL

In addition to communicating over ProfiBus, commands in the Class 5 SmartMotor™ ASCII language may be sent through the channel 0 RS232 port or through the channel 1 RS485 port.

Reserved Motor Variables:

The Class 5 Profibus interface no longer reserves user variables yyy, and zzz as the Class 4 did. These variables are freely available for the user.

The Class 5 Profibus interface no longer uses the channel 1 RS-485 serial port. Therefore, that port is freely available to the user for any application.

The Class 5 Profibus interface no longer uses the EPTR variable of the EEPROM command set. When the Profibus accesses the EEPROM, it is done through its own private version of EPTR. Therefore, the user no longer has to monitor variable zzz for shared access. The user may access the EEPROM at any time. EEPROM writes are buffered internally. EEPROM reads may still cause a user command to wait until the EEPROM is available, but this is automatic to the user.

APPENDIX:

ProfiBus Packet Command and Response Codes

Command Packet Command Codes to Motor Commands Class 5

Commands that have different meanings in different motor versions, such as RBA, will have the meaning associated with the version of the motor.

Variables beginning with u8, u16, or u32 are internal to the motor's ProfiBus module.

Command Code	Command Data Value	Note	Description	SmartMotor Command(s)	SmartMotor Response
decimal,hex	decimal,hex				
0		NULL	no command		
1, x01	0, x00		engage brake	BRKENG	
1, x01	1, x01		Use only internal brake. Disable external brake.	EOBK(-1)	
1, x01	2, x02		Direct brake to output pin 6 (port G)	EOBK(6)	
1, x01	3, x03		Direct brake to output pin 2 (port C)	EOBK(2)	
1, x01	4, x04		release brake	BRKRLS	
1, x01	5, x05		brake while servo inactive	BRKSRV	
1, x01	6, x06		brake while trajectory inactive	BRKTRJ	
1, x01	7, x07		Not implemented.		
1, x01	8, x08		select internal encoder for servo	ENC0	
1, x01	9, x09		select external encoder for servo	ENC1	
1, x01	10, x0A		End user program	END	
1, x01	11, x0B		Transfer buffered PID tuning to live values.	F	
1, x01	12, x0C		start motion (GO)	G	
1, x01	13, x0D		PID gravity mode off (default state)	KGOFF	
1, x01	14, x0E		PID gravity mode on	KGON	
1, x01	15, x0F		Obsolete, don't use		
1, x01	16, x10		Obsolete, don't use		
1, x01	17, x11		Obsolete, don't use		
1, x01	18, x12		Obsolete, don't use		
1, x01	19, x13		CAM mode - not implemented	MC	
1, x01	20, x14		Obsolete, don't use.		
1, x01	21, x15		Obsolete, don't use.		
1, x01	22, x16		Obsolete, don't use.		
1, x01	23, x17		enable contouring mode – not implemented.	MD	
1, x01	24, x18		set mode follow, and zero out	MF0	
1, x01	25, x19		Obsolete, don't use.		
1, x01	26, x1A		Obsolete, don't use.		
1, x01	27, x1B		Obsolete, don't use.		
1, x01	28, x1C		initiate mode follow quadrature	MFR	
1, x01	29, x1D		enable position mode	MP	
1, x01	30, x1E		Obsolete, don't use.		
1, x01	31, x1F		configure step and direction, and zero out	MS0	

1, x01	32, x20		initate mode step ratio calculation	MSR	
1, x01	33, x21		enable torque mode	MT	
1, x01	34, x22		Immediately engage MTB brake.	MTB	
1, x01	35, x23		enable velocity mode	MV	
1, x01	36, x24		stop servoing the motor	OFF	
1, x01	37, x25		divide PID sample rate by 1	PID1	
1, x01	38, x26		divide PID sample rate by 2, default	PID2	
1, x01	39, x27		divide PID sample rate by 4	PID4	
1, x01	40, x28		divide PID sample rate by 8	PID8	
1, x01	41, x29		execute stored program	RUN	
1, x01	42, x2A		End program if RUN has not been commanded yet since power up	RUN?	
1, x01	43, x2B		stop move in progress abruptly	S	
1, x01	44, x2C		Make IO 0 (port A) an input. If a brake redirect, this is ignored.	EIGN(0)	
1, x01	45, x2D		Obsolete. Use instead: CMD_OUT(x)		
1, x01	46, x2E		Make IO 1 (port B) an input. If a brake redirect, this is ignored.	EIGN(1)	
1, x01	47, x2F		Obsolete. Use instead: CMD_OUT(x)		
1, x01	48, x30		Make IO 2 (port C) an input. If a brake redirect, this is ignored.	EIGN(2)	
1, x01	49, x31		Obsolete. Use instead: CMD_OUT(x)		
1, x01	50, x32		set I/O C to be a right limit input	EILP	
1, x01	51, x33		Make IO 3 (port D) an input. If a brake redirect, this is ignored.	EIGN(3)	
1, x01	52, x34		Obsolete. Use instead: CMD_OUT(x)		
1, x01	53, x35		set I/O D to be a left limit input	EILN	
1, x01	54, x36		slow motor motion to stop	X	
1, x01	55, x37		total system reset	Z	
1, x01	56, x38		reset current limit violation latch bit	Za	
1, x01	57, x39		reset serial data parity violation latch bit	Zb	
1, x01	58, x3A		reset communications buffer overflow latch bit	Zc	
1, x01	59, x3B		Available in Class 4 emulation mode only to reset math error.	Zd	
1, x01	60, x3C		Reset position error fault.	Ze	
1, x01	61, x3D		reset serial comm framing error latch bit	Zf	
1, x01	62, x3E		Reset over-temperature fault, requires temperature to fall 5 degrees below limit.	Zh	
1, x01	63, x3F		reset historical left limit latch bit	Zl	
1, x01	64, x40		reset historical right limit latch bit	Zr	
1, x01	65, x41		reset command scan error latch bit	Zs	
1, x01	66, x42		Available in Class 4 emulation mode only to reset array index error.	Zu	
1, x01	67, x43		reset encoder wrap around event latch bit	Zw	
1, x01	68, x44		reset system latches to power-up state	ZS	
1, x01	69, x45		disable software limits	SLD	
1, x01	70, x46		enable software limits	SLE	
1, x01	71, x47		Make IO 6 (port G) an input. If a brake redirect, this is ignored. Disable GO synchronization function	EIGN(6)	
1, x01	72, x48		enable GO synchronization function	EISM(6)	

1, x01	73, x49		Make IO 4 (port E) an input. If a brake redirect, this is ignored.	EIGN(4)	
1, x01	74, x4A		Make IO 5 (port F) an input. If a brake redirect, this is ignored.	EIGN(5)	
1, x01	75, x4B		Arm index capture from internal encoder rising edge.	Ai(0)	
1, x01	76, x4C		Arm index capture from internal encoder falling edge.	Aj(0)	
1, x01	77, x4D		Arm index capture from internal encoder rising then falling edge.	Aij(0)	
1, x01	78, x4E		Arm index capture from internal encoder falling then rising edge.	Aji(0)	
1, x01	79, x4F		Arm index capture from external encoder rising edge.	Ai(1)	
1, x01	80, x50		Arm index capture from external encoder falling edge.	Aj(1)	
1, x01	81, x51		Arm index capture from internal encoder rising then falling edge.	Aij(1)	
1, x01	82, x52		Arm index capture from internal encoder falling then rising edge.	Aji(1)	
1, x01	83, x53		Immediately force trapezoidal-hall commutation.	MDT	
1, x01	84, x54		Request enhanced trapezoidal-encoder commutation. Mode entered as soon as angle is satisfied.	MDE	
1, x01	85, x55		Request sine voltage mode commutation. Mode entered as soon as angle is satisfied.	MDS	
1, x01	86, x56	Reserved	Reserved	MDC	
1, x01	87, x57		Turn on TOB-feature for trapezoidal mode.	MDB	
1, x01	88+, x58+	unused			
2, x02	<value>	DO_MOVE_POS_ABS	set absolute position and start motor	PT=<value> G	
3, x03	<value>	DO_MOVE_POS_REL	set relative position and start motor	PRT=<value> G	
4, x04	<value>	DO_MOVE_VEL	Set velocity and start motor	VT=<value> G	
5, x05	<value>		call a subroutine	GOSUB(<value>)	
6, x06	<value>		branch program execution to a label	GOTO(<value>)	
7, x07		Reserved	Not implemented.		
8, x08		Reserved	Not implemented.		
9-89, x09-x59		unused			
90, x5A	<value>		Clear mask on user bits, word 0. Status word 12.	UR(W,0,<value>)	
91, x5B	<value>		Clear mask on user bits, word 1. Status word 13.	UR(W,1,<value>)	
92, x5C	<value>		Set mask on user bits, word 0. Status word 12.	US(W,0,<value>)	
93, x5D	<value>		Set mask on user bits, word 1. Status word 13.	US(W,1,<value>)	
94, x5E	<value>		Clear specific user bit 0-31	UR(<value>)	
95, x5F	<value>		Set specific user bit 0-31	US(<value>)	
96, x60	<value>		Set digital output 4 (port e) to a 0 or a 1.	OUT(4)=<value>	
97, x61	<value>	Unused			
98, x62	<value>		Set digital output 5 (port f) to a 0 or a 1.	OUT(5)=<value>	
99, x63	<value>	Unused			
100, x64	<value>		set acceleration	ADT=<value>	
101, x65	<value>			ADDR=<value>	
102, x66	<value>		set PWM drive signal limit	AMPS=<value>	
103, x67		Reserved	Not implemented.		
104-123, x68-x7B		unused			

124, x7C	<value>		set relative distance (position)	PRT=<value>	
125, x7D	<value>		set allowable position error	EL=<value>	
126, x7E		unused			
127, x7F			Obsolete		
128, x80		unused			
129, x81	<value>		PID acceleration feed forward	KA=<value>	
130, x82	<value>		PID derivative compensation	KD=<value>	
131, x83	<value>		PID gravity compensation	KG=<value>	
132, x84	<value>		PID integral compensation	KI=<value>	
133, x85	<value>		PID integral limit	KL=<value>	
134, x86	<value>		PID proportional compensation	KP=<value>	
135, x87	<value>		PID derivative term sample rate	KS=<value>	
136, x88	<value>		PID velocity feed forward	KV=<value>	
137, x89	<value>		mode follow with ratio divisor	MFDIV=<value>	
138, x8A	<value>		mode follow with ratio multiplier	MFMUL=<value>	
139, x8B	<value>		set origin	O=<value>	
140, x8C	<value>		shift origin	OSH(<value>)	
141, x8D		unused			
142, x8E	<value>		set absolute position target	PT=<value>	
143-144, x8F-x90		unused			
145, x91	<value>			SADDR<value>	
146, x92		Reserved	Not implemented		
147, x93		Unused			
148, x94	<value>		assign torque value in torque mode	T=<value>	
149, x95		Unused			
150, x96	<value>		set high temperature setpoint	TH=<value>	
151, x97	<value>		set temperature fault delay	THD=<value>	
152, x98	<value>		set I/O A output	OUT(0)=<value>	
153, x99		Unused			
154, x9A	<value>		set I/O B output	OUT(1)=<value>	
155, x9B		Unused			
156, x9C	<value>		set I/O C output	OUT(2)=<value>	
157, x9D		Unused			
158, x9E	<value>		set I/O D output	OUT(3)=<value>	
159, x9F		Unused			
160, xA0	<value>		set I/O G output	OUT(6)=<value>	
161-162, xA1-xA2		Unused			
163, xA3	<value>		set maximum permitted velocity	VT=<value>	
164, xA4		Reserved			
165, xA5	<value>		set value of negative software limit	SLN=<value>	
166, xA6	<value>		set value of positive software limit	SLP=<value>	
167-169, xA7-xA9		Unused			
170, xAA	<value>		Clear status word 0, bit indicated by value.	Z(0,<value>)	
171, xAB	<value>		Clear status word 1, bit indicated by value.	Z(1,<value>)	
172, xAC	<value>		Clear status word 2, bit indicated by value.	Z(2,<value>)	

173, xAD	<value>		Clear status word 3, bit indicated by value.	Z(3,<value>)	
174, xAE	<value>		Clear status word 4, bit indicated by value.	Z(4,<value>)	
175, xAF	<value>		Clear status word 5, bit indicated by value.	Z(5,<value>)	
176, xB0	<value>		Clear status word 6, bit indicated by value.	Z(6,<value>)	
177, xB1	<value>		reserved for status words.		
178, xB2	<value>		reserved for status words.		
179, xB3	<value>		reserved for status words.		
180, xB4	<value>		reserved for status words.		
181, xB5	<value>		reserved for status words.		
182, xB6	<value>		reserved for status words.		
183, xB7	<value>		reserved for status words.		
184, xB8	<value>		reserved for status words.		
185, xB9	<value>		reserved for status words.		
186, xBA	<value>		reserved for status words.		
187, xBB	<value>		reserved for status words.		
188, xBC	<value>		reserved for status words.		
189, xBD	<value>		reserved for status words.		
190-199, xBE-xC7		Unused			
200, C8	<value> 0-25	SET_VAR_INDEX_SET	u8VarIndexSet = <value> u8VarIndexSetActual = <value>		
201, xC9		Unused			
202, xCA	<value> 0-26	SET_VAR_LEN_SET	u8VarLenSet = <value>		
203, xCB	<value> 0-199	SET_ARRAY_INDEX_SET	u8ArrIndexSet = <value> u8ArrIndexSetActual = <value>		
204, xCC		unused			
205, xCD	<value> 0-200	SET_ARR_LEN_SET	u8ArrLenSet = <value>		
206, xCE	<value> 0=NO 1=YES	SET_AUTO_INC_SET	u8AutoIncSet = <value>		
207, xCF	<value> 0-25	SET_VAR_INDEX_GET	u8VarIndexGet = <value> u8VarIndexGetActual = <value>		
208, xD0		unused			
209, xD1	<value> 0-26	SET_VAR_LEN_GET	u8VarLenGet = <value>		
210, xD2	<value> 0-199	SET_ARRAY_INDEX_GET	u8ArrIndexGet = <value> u8ArrIndexGetActual = <value>		
211, xD3		unused			
212, xD4	<value> 0-200	SET_ARR_LEN_GET	u8ArrLenGet = <value>		
213, xD5	<value> 0=NO 1=YES	SET_AUTO_INC_GET	u8AutoIncGet = <value>		
214, xD6	<value>	SET_VAR	set variable <a to z>='a'+u8VarIndexSetActual If (u8AutoIncSet) then u8VarIndexSetActual += 1	<a to z>= <value>	

215, xD7	<value>	SET_ARRAY_BYTE	set byte array variable <index>=u8ArrIndexSetActual If (u8AutoIncSet) then u8ArrIndexSetActual += 1	ab[<index>]= <value>	
216, xD8	<value>	SET_ARRAY_WORD	set word array variable <index>=u8ArrIndexSetActual If (u8AutoIncSet) then u8ArrIndexSetActual += 1	aw[<index>]= <value>	
217, xD9	<value>	SET_ARRAY_LONG	set long array variable <index>=u8ArrIndexSetActual If (u8AutoIncSet) then u8ArrIndexSetActual += 1	al[<index>]= <value>	
218, xDA	<value>	SET_NVOL_BYTE	store byte to eeprom u32EptrActual += 1	VST(<value byte>,1)	
219, xDB	<value>	SET_NVOL_WORD	store word to eeprom u32EptrActual += 2	VST(<value word16>,1)	
220, xDC	<value>	SET_NVOL_LONG	store long to eeprom u32EptrActual += 4	VST(<value long>,1)	
221, xDD	<value>	SET_NVOL_VAR	set variable and store to eeprom <a to z>='a'+u8VarIndexSetActual u32EptrActual += 4 If (u8AutoIncSet) then u8VarIndexSetActual += 1	<a to z>=<value> VST(<a to z>,1)	
222, xDE		STORE_NVOL_VARS	store vars to eeprom <a to z>='a'+u8VarIndexSetActual <length>=u8VarLenSet u32EptrActual += (<length>*4) If (u8AutoIncSet) then u8VarIndexSetActual += <length>	VST(<a to z>, <length>)	
223, xDF		STORE_NVOL_ARRAY_BYTE	store byte array variables to eeprom <index>=u8ArrIndexSetActual <length>=u8ArrLenSet u32EptrActual += (<length>*1) If (u8AutoIncSet) then u8ArrIndexSetActual += <length>	VST(ab[<index>], <length>)	
224, xE0		STORE_NVOL_ARRAY_WORD	store word array variables to eeprom <index>=u8ArrIndexSetActual <length>=u8ArrLenSet u32EptrActual += (<length>*2) If (u8AutoIncSet) then u8ArrIndexSetActual += <length>	VST(aw[<index>], <length>)	
225, xE1		STORE_NVOL_ARRAY_LONG	store long array variables to eeprom <index>=u8ArrIndexSetActual <length>=u8ArrLenSet u32EptrActual += (<length>*4) If (u8AutoIncSet) then u8ArrIndexSetActual += <length>	VST(al[<index>],<length>)	
226, xE2	<value>		set the eeprom address u32EptrSet=<value> u32EptrActual=<value> (doesn't affect EPTR)		
227, xE3		unused			

228, xE4	<value> 3-65535	SET_NET_LOST_LABEL	set goto/gosub number for net lost action u16NetLostLabel=<value> (initialized to the value of u16NetLostLabelDefault during power-up)		
229, xE5	<value>	SET_NET_LOST_ACTION	set ProfiBus network lost action u8NetLostAction=<value> (initialized to the value of u8NetLostActionDefault during power-up)	Upon loss of comm with ProfiBus host, command is based on <value>: 0=IGNORE (No Command), 1=OFF (Motor Off), 2=X (Soft Stop), 3=S (Immediate Stop), 4=GOSUB, 5=GOTO	
230, xE6	<value> 0-65000	SET_POLL_RATE	set rate (in us) at which the SmartMotor is polled by the ProfiBus Gateway u32PollRate=<value> (initialized to the value read from SmartMotor EEPROM location 32012 during power up)		
231, xE7	<value> 3-65535	SET_NET_LOST_LABEL_DEFAULT	set goto/gosub number for net lost action u16NetLostLabelDefault=<value> (saved in non-volatile storage on the ProfiBus gateway)		
232, xE8	<value>	SET_NET_LOST_ACTION_DEFAULT	set ProfiBus network lost action u8NetLostActionDefault=<value> (saved in non-volatile storage on the ProfiBus gateway)	Upon loss of comm with ProfiBus host, command is based on <value>: 0=IGNORE (No Command), 1=OFF (Motor Off), 2=X (Soft Stop), 3=S (Immediate Stop), 4=GOSUB, 5=GOTO	
233, xE9	<value>	SET_NET_DEVICE_ID	u32NetDeviceId=<value> (saved in non-volatile storage on the ProfiBus gateway)	CADDR=<value>	
234, xEA			Not implemented		
235-239, xEB-xEF		Unused			
240, xF0	<value>	SET_PA_FIELD	Configure the Profibus input packet to use an alternate data source for the 'Measured position' field (words 4,5). <value>: 0 - report actual position in encoder counts. This is the power-up default value. 1 - report al[0] (big-end format.) 2 - report af[0] (IEEE-754 32-bit single precision. Big-end format.)	CANCTL(10,x)	
241-253, xF1-xFD		Unused			
254, xFE		CMD_RESTORE_DEFAULTS	Restores all non-volatile Gateway settings to defaults. (u16NetLostLabelDefault=3 u8NetLostAction=IGNORE u32NetDeviceId=0 u32PollRate=0)		
255, xFF		ERROR	returned if the command code could not be performed successfully		

**Response Packet Codes to Motor Commands
Class 5**

Commands that have different meanings in different motor versions, such as RBa, will have the meaning associated with the version of the motor.

Variables beginning with u8, u16, or u32 are internal to the motor's ProfiBus module.

Response Code	Response Data Value	Mnemonic	Description	SmartMotor Command(s)	SmartMotor Response
decimal,hex					
0, x00	0	NULL	no command		
1-95, x01-x5f		unused			
96, x60	<value>		Report digital IO pin 4 (port E.)	RIN(4)	
97, x61	<value>		Report analog input pin 4 (port E.)	RINA(A,4)	
98, x62	<value>		Report digital IO pin 5 (port F.)	RIN(5)	
99, x63	<value>		Report analog input pin 5 (port F.)	RINA(A,5)	
100, x64	<value>		report acceleration	RAT	<value>
101, x65	<value>		get smartmotor address	RADDR	<value>
102, x66	<value>		report assigned max drive PWM limit	RAMPS	<value>
103, x67	<value>		report over current status	RBa	<value>
104, x68	<value>		Obsolete, don't use.		
105, x69	<value>		Obsolete, don't use.		
106, x6A	<value>		Obsolete, don't use.		
107, x6B	<value>		report position error status	RBe	<value>
108, x6C	<value>		Obsolete, don't use.		
109, x6D	<value>		report overheat status	RBh	<value>
110, x6E	<value>		report index status	RBi	<value>
111, x6F	<value>		Obsolete, don't use.		
112, x70	<value>		report historical left limit status	RBl	<value>
113, x71	<value>		report negative limit status	RBm	<value>
114, x72	<value>		report motor off status	RBo	<value>
115, x73	<value>		report positive limit status	RBp	<value>
116, x74	<value>		report historical right limit status	RBr	<value>
117, x75	<value>		report program scan status	RBs	<value>
118, x76	<value>		report trajectory status	RBt	<value>
119, x77	<value>		Obsolete, don't use.		
120, x78	<value>		report wrap around status	RBw	<value>
121, x79	<value>		report hardware index input level	RBx	<value>
122, x7A	<value>		Report millisecond clock.	RCLK	<value>
123, x7B	<value>		report secondary counter	RCTR(1)	<value>
124, x7C	<value>		report buffered move distance value	RPRC	<value>
125, x7D	<value>		report buffered maximum position error	REL	<value>
126, x7E		unused			
127, x7F	<value>		Obsolete, don't use.		

128, x80	<value>		Report index position captured from recent Ai(0) command - object 1, data value 75. When class 4 emulation, this re-arms the capture.	RI(0)	<value>
129, x81	<value>		report buffered acceleration feed forward coefficient	RKA	<value>
130, x82	<value>		report buffered derivative coefficient	RKD	<value>
131, x83	<value>		report buffered gravity coefficient	RKG	<value>
132, x84	<value>		report buffered integral coefficient	RKI	<value>
133, x85	<value>		report buffered integral limit	RKL	<value>
134, x86	<value>		report buffered proportional coefficient	RKP	<value>
135, x87	<value>		report buffered sampling interval	RKS	<value>
136, x88	<value>		report buffered velocity feed forward coefficient	RKV	<value>
137, x89	<value>		report follow mode divisor	RMFDIV	<value>
138, x8A	<value>		report follow mode multiplier	RMFMUL	<value>
139, x8B		unused			
140, x8C	<value>		report current mode of operation	RMODE	<value>
141, x8D	<value>		report present position	RPA	<value>
142, x8E	<value>		report buffered position setpoint	RPT	<value>
143, x8F	<value>		report present position error	REA	<value>
144, x90			Not implemented.		
145-146, x91-x92		unused			
147, x93			Not implemented.		
148, x94	<value>		report current requested torque	RT	<value>
149, x95	<value>		report temperature	RTEMP	<value>
150, x96	<value>		Report temperature shutdown limit	RTH	<value>
151, x97	<value>		Report current algorithm THD time.	RTHD	<value>
152, x98	0=Bit 0 Off 1=Bit 0 On		Report digital IO pin 0 (port A.)	RIN(0)	<value>
153, x99	<value>		Report analog input pin 0 (port A.)	RINA(A,0)	<value>
154, x9A	0=Bit 1 Off 1=Bit 1 On		Report digital IO pin 1 (port B.)	RIN(1)	<value>
155, x9B	<value>		Report analog input pin 1 (port B.)	RINA(A,1)	<value>
156, x9C	0=Bit 2 Off 1=Bit 2 On		Report digital IO pin 2 (port C.)	RIN(2)	<value>
157, x9D	<value>		Report analog input pin 2 (port C.)	RINA(A,2)	<value>
158, x9E	0=Bit 3 Off 1=Bit 3 On		Report digital IO pin 3 (port D.)	RIN(3)	<value>
159, x9F	<value>		Report analog input pin 3 (port D.)	RINA(A,3)	<value>
160, xA0	0=Bit 6 Off 1=Bit 6 On		Report digital IO pin 6 (port G.)	RIN(6)	<value>
161, xA1	<value>		Report analog input pin 6 (port G.)	RINA(A,6)	<value>
162, xA2	<value>		report velocity	RVA	<value>
163, xA3	<value>		report buffered maximum velocity	RVT	<value>
164, xA4	<value>		report Legacy status word	(n/a)	<value>
165, xA5	<value>		value of negative software limit	RSLN	<value>

166, xA6	<value>		value of positive software limit	RSLP	<value>
167, xA7	<value>		Not implemented.		
168, xA8	<value>		pin A B C D E F G 7bit value	RIN(W,0)	<value>
169, xA9		unused			
170, xAA	<value>		report status word 0	RW(0)	<value>
171, xAB	<value>		report status word 1	RW(1)	<value>
172, xAC	<value>		report status word 2	RW(2)	<value>
173, xAD	<value>		report status word 3	RW(3)	<value>
174, xAE	<value>		report status word 4	RW(4)	<value>
175, xAF	<value>		report status word 5	RW(5)	<value>
176, xB0	<value>		report status word 6	RW(6)	<value>
177, xB1	<value>		reserved		
178, xB2	<value>		reserved		
179, xB3	<value>		reserved		
180, xB4	<value>		reserved		
181, xB5	<value>		reserved		
182, xB6	<value>		report user bits 0-15 (Status word 12)	RW(12)	
183, xB7	<value>		report user bits 16-31 (Status word 13)	RW(13)	
184, xB8	<value>		reserved		
185, xB9	<value>		reserved		
186, xBA	<value>		report I/O 0-7 (Status word 16)	RW(16)	
187, xBB	<value>		reserved		
188, xBC	<value>		reserved		
189, xBD	<value>		reserved		
190-199, xBE-xC7		unused			
200, xC8	<value>	GET_VAR_INDEX_SET	<value>=u8VarIndexSet		
201, xC9	<value>	GET_VAR_INDEX_SET_ACTUAL	<value>=u8VarIndexSetActual		
202, xCA	<value>	GET_VAR_LEN_SET	<value>=u8VarLenSet		
203, xCB	<value>	GET_ARRAY_INDEX_SET	<value>=u8ArrIndexSet		
204, xCC	<value>	GET_ARRAY_INDEX_SET_ACTUAL	<value>=u8ArrIndexSetActual		
205, xCD	<value>	GET_ARR_LEN_SET	<value>=u8ArrLenSet		
206, xCE	<value>	GET_AUTO_INC_SET	<value>=u8AutoIncSet		
207, xCF	<value>	GET_VAR_INDEX_GET	<value>=u8VarIndexGet		
208, xD0	<value>	GET_VAR_INDEX_GET_ACTUAL	<value>=u8VarIndexGetActual		
209, xD1	<value>	GET_VAR_LEN_GET	<value>=u8VarLenGet		
210, xD2	<value>	GET_ARRAY_INDEX_GET	<value>=u8ArrIndexGet		
211, xD3	<value>	GET_ARRAY_INDEX_GET_ACTUAL	<value>=u8ArrIndexGetActual		
212, xD4	<value>	GET_ARR_LEN_GET	<value>=u8ArrLenGet		
213, xD5	<value>	GET_AUTO_INC_GET	<value>=u8AutoIncGet		
214, xD6	<value>	GET_VAR	get variable <a to z>='a'+u8VarIndexGetActual If (u8AutoIncGet) then u8VarIndexGetActual += 1	R<a to z> (Issued Only One Time)	<value>

215, xD7	<value>	GET_ARRAY_BYTE	get byte array variable <index>=u8ArrIndexGetActual If (u8AutoIncGet) then u8ArrIndexGetActual += 1	Rab[<index>] (Issued Only One Time)	<value>
216, xD8	<value>	GET_ARRAY_WORD	get word array variable <index>=u8ArrIndexGetActual If (u8AutoIncGet) then u8ArrIndexGetActual += 1	Raw[<index>] (Issued Only One Time)	<value>
217, xD9	<value>	GET_ARRAY_LONG	get long array variable <index>=u8ArrIndexGetActual If (u8AutoIncGet) then u8ArrIndexGetActual += 1	Ral[<index>] (Issued Only One Time)	<value>
218, xDA	<value>	GET_NVOL_BYTE	get byte from eeprom u32EptrActual += 1	VLD(<value>,1) (Issued Only One Time)	<value>
219, xDB	<value>	GET_NVOL_WORD	get word from eeprom u32EptrActual += 2	VLD(<value>,1) (Issued Only One Time)	<value>
220, xDC	<value>	GET_NVOL_LONG	get long from eeprom u32EptrActual += 4	VLD(<value>,1) (Issued Only One Time)	<value>
221, xDD	<value>	GET_NVOL_VAR	get variable from eeprom <a to z>='a'+u8VarIndexGetActual u32EptrActual += 4 If (u8AutoIncGet) then u8VarIndexGetActual += 1	VLD(<a to z>,1) R<a to z> (Issued Only One Time)	<value>
222, xDE		LOAD_NVOL_VARS	load vars from eeprom <a to z>='a'+u8VarIndexGetActual <length>=u8VarLenGet u32EptrActual += (<length>*4) If (u8AutoIncGet) then u8VarIndexGetActual += <length>	VLD(<a to z>,<length>) (Issued Only One Time)	
223, xDF		LOAD_NVOL_ARRAY_BYTE	load byte array variables from eeprom <index>=u8ArrIndexGetActual <length>=u8ArrLenGet u32EptrActual += (<length>*1) If (u8AutoIncGet) then u8ArrIndexGetActual += <length>	VLD(ab[<index>],<length>) (Issued Only One Time)	
224, xE0		LOAD_NVOL_ARRAY_WORD	load word array variables from eeprom <index>=u8ArrIndexGetActual <length>=u8ArrLenGet u32EptrActual += (<length>*2) If (u8AutoIncGet) then u8ArrIndexGetActual += <length>	VLD(aw[<index>],<length>) (Issued Only One Time)	
225, xE1		LOAD_NVOL_ARRAY_LONG	load long array variables from eeprom <index>=u8ArrIndexGetActual <length>=u8ArrLenGet u32EptrActual += (<length>*4) If (u8AutoIncGet) then u8ArrIndexGetActual += <length>	VLD(al[<index>],<length>) (Issued Only One Time)	
226, xE2	<value>		get last set eeprom address <value>=u32EptrSet		

227, xE3	<value>		get actual eeprom address setting in Profi interface <value>=u32EptrActual		
228, xE4	<value>	GET_NET_LOST_LABEL	<value>=u16NetLostLabel (initialized to the value of u16NetLostLabelDefault during power-up)		
229, xE5	<value>	GET_NET_LOST_ACTION	<value>=u8NetLostAction (initialized to the value of u8NetLostActionDefault during power-up)	Upon loss of comm with ProfiBus host, command is based on <value>: 0=IGNORE (No Command), 1=OFF (Motor Off), 2=X (Soft Stop), 3=S (Immediate Stop), 4=GOSUB, 5=GOTO	
230, xE6	<value>	GET_POLL_RATE	<value>=u32PollRate Rate at which SmartMotor is polled by the ProfiBus gateway. (initialized to the value read from SmartMotor EEPROM location 32012 during power up)		
231, xE7	<value>	GET_NET_LOST_LABEL_DEFAULT	<value>=u16NetLostLabelDefault (saved in non-volatile storage on the ProfiBus gateway)		
232, xE8	<value>	GET_NET_LOST_ACTION_DEFAULT	<value>=u8NetLostActionDefault (saved in non-volatile storage on the ProfiBus gateway)	Upon loss of comm with ProfiBus host, command is based on <value>: 0=IGNORE (No Command), 1=OFF (Motor Off), 2=X (Soft Stop), 3=S (Immediate Stop), 4=GOSUB, 5=GOTO	
233, xE9	<value>	GET_NET_DEVICE_ID	<value>=u32NetDeviceId (saved in non-volatile storage on the ProfiBus gateway)	RCADDR	
234, xEA			Not implemented.		
235, xEB	<value>	GET_ENC_RESOLUTION	<value>=s32EncResolution (read at startup from SmartMotor EEPROM location 32000)	RRES	
236, xEC	<value>	GET_FIRMWARE_VERSION	<value>=u32FirmwareVersion (read at startup w/ RSP)	RFW	
237, xED			Not implemented.		
238, xEE	<value>	GET_SAMPLE_RATE	<value>=s32SampleRate (read at startup w/ RSP)		
239, xEF	<value>	GET_MOTOR_ID	<value>=u32MotorId (read at startup from SmartMotor EEPROM location 32016)		
240-254, xF0-xFE		Unused			
255, xFF	<error code>	ERROR	returned if the response code could not be performed successfully (<error code> values to be determined)		

Troubleshooting

No ProfiBus Connection

The ProfiBus connection status LED is off after power up and while connected to the master.

Background information:

Gateway auto-detect of master's ProfiBus baud rate may take 5 to 10 seconds after motor power up. ProfiBus master repeatedly sends the command packet "are you there? ... are you there?" Slave motor does not transmit on ProfiBus unless commanded to answer by the master.

Check the following:

- motor power is on
- connections made between Profi master and slave motor
- motor non-volatile settings
 - Motor MAC ID has been set to the value the host PLC is expecting.
 - Polling rate is set to 0 to allow fastest response.
- terminator and bias resistors are correct:

using Siemen's connectors:

- switch on Siemens Profi connector is in "on" position at both ends of bus (for example at Master and slave motor if that is the whole bus)
- switch on Siemens Profi connector is in "off" position at all other nodes, exactly two are on

not using Siemen's connectors:

- see ProfiBus Connector Pinouts>DSUB below for bias and terminator resistor
- values, total termination resistance is 110 ohms
- cable is correct: cable is ProfiBus cable
- oscilloscope shows good square 3v signals
- ProfiBus A and B signals are not reversed
- master was configured using GSD file DEAF070C.GSD when defining the slave motor as part of the ProfiBus network

ProfiBus LED

No Power to Motor - Profibus LED is off. The 'power' LED on top of the motor is also off.

After Motor Power-Up ProfiBus Gateway Initialization

Profibus "Connection Status" LED

- | | |
|--------------|---|
| OFF | no connection to ProfiBus Host, or watchdog expired. Verify that host is communicating to the MAC ID set for this motor. |
| GREEN | connection to ProfiBus Host has been established if Master has enabled and initialized slave (Motor) watchdog timer. Host continues to send traffic to motor to keep watchdog active. |