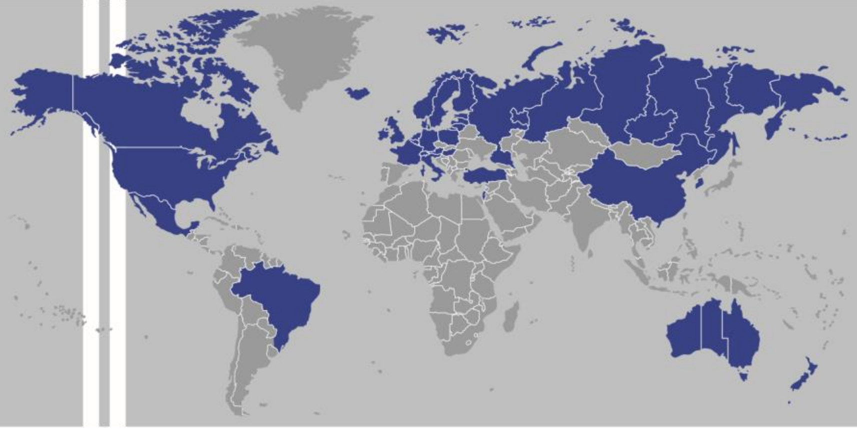

dunkermotoren
advanced motion solutions

DME 230 / 400

Typ:	Part No:
DME 230x4-PN	81703.00103
DME 400x8-PN	81703.00112

Profinet Interface

for servo drives series

- BN6773
- BN6783

Publication Ref: 160616

Dunkermotoren GmbH | Allmendstraße 11 | D-79848 Bonndorf/ Schwarzwald
Phone +49 (0) 7703 930-0 | Fax +49 (0) 7703 930-210/ 212 | info@dunkermotoren.com

AMETEK®
PRECISION MOTION CONTROL

Servo Drives New Generation

Digital Servo Drives
for Direct Mains Connection

Profinet Interface

Operating Instructions 6770.237, V 1.0

These operating instructions apply to

- Servo Drives New Generation, compact design, with integrated safety system
  - BN 6771 to BN 6774 with built-in power supply unit for single-phase AC power connection
  - BN 6781 to BN 6787 with built-in power supply unit for three-phase power connection
- Operation via personal computer with SPP Windows software
- Access to device functions via communication interfaces

These operating instructions are applicable together with

- Operating Instructions 6710.201 (Functions and Parameters)*
- Operating Instructions 6770.202 (Connection and Commissioning)
- Operating Instructions 6710.207 (SPP Windows Command and Commissioning Software)*

* Available in the help function of SPP Windows and per download

ESR Pollmeier GmbH
Lindenstr. 20

64372 Ober-Ramstadt

Federal Republic of Germany

Phone +49 6167 9306-0
Fax +49 6167 9306-77

E-mail info@esr-pollmeier.de
www.esr-pollmeier.de

Versions of the Document

2015-09-14	V 1.0, KS	new on the basis of the German version

Contents

Please, also refer to the index at the end of this document.

1 Preliminary Remarks

1.1 About this Description

These Operating Instructions 6770.237 explain the Profinet interface of the digital Servo Drives New Generation.

They are applicable together with

- Operating Instructions “Connection and Commissioning” of the servo drive (included in the scope of delivery of the servo drive)
  - Operating Instructions 6770.202 (Servo Drives New Generation)
- Operating Instructions “Functions and Parameters” of the servo drive (available in the SPP Windows help function and per download)
  - Operating Instructions 6710.201

as well as, depending on the equipment,

- Operating Instructions “SPP Windows Command and Commissioning Software” (available in the SPP Windows help function and per download)
  - Operating Instructions 6710.207
- Operating Instructions “Part Program” (running motion sequences independent of a higher-level controller; available in the SPP Windows help function and per download)
  - Operating Instructions 6710.231

The following documents are additionally required for working with the Profinet interface:

- operating instructions of the Profinet modules used for the controller and the corresponding software (IO controller) as well as
- a configuration tool compatible with the IO controller.

Before accessing the servo drives via the Profinet interface according to these operating instructions, put the servo drive system (servo drive and servo motor) into operation. For that, a PC with command and commissioning software SPP Windows is required. Please, make sure that these requirements are met.

1.2 Function Blocks

Function blocks are available for an easy integration of the servo drive systems into automation systems.

These are available for Siemens Simatic S7 and various controllers according to IEC 61131-3. For further information, please contact ESR.

Communication takes place via Profinet IO.

Supported functions:

- parameterization of the servo drive systems by the controller (e. g. after switch-on)

- triggering of movements (relative/absolute positioning, going to home position, speed setting, ...)
- influencing the positioning control (part program) integrated in the drive system
- input and output of binary signals (software inputs/outputs)
- example programs for using the function library as a basis for developing own programs

The function blocks are based on PLCopen specification “Function Blocks for Motion Control” which is based on IEC 61131-3.

For further information see data sheet 6710.260.

The function blocks simplify the use of many functions described in these operating instructions. The parameterization steps you have to carry out yourself to parameterize the Profinet interface are described in the operating instructions of the function blocks.

2 Safety Instructions

In any case, observe the safety instructions in operating instructions “Connection and Commissioning” (6770.202) as well as the warnings and hints in the margins of all operating instructions.

Access to the servo drives via Profinet interface may trigger drive system movements. If drive system and/or machine have not been set up and secured properly, health and life of persons may be endangered.

Therefore, access via Profinet interface is prohibited until the requirements of the machinery directive have been met.

In bus systems, a bus participant can be influenced invisibly from outside. This can lead to an unexpected (uncontrollable) system behavior. Do not put the bus into operation unless you have made sure that all participants are properly connected and configured.

2.1 Type of Instructions

The warnings and hints in the margin must be observed in any case:

- **Danger** to health and life due to electrical shock or motion of the drive.
- **Caution**: Noncompliance violates the safety regulations or statutory provisions and can lead to personal injury or material damage.
- **Check**: Prior to commissioning and in case of failures or problems, check these items first.
- **Tip**, useful hint

3 Technical Specifications

The Profinet interface is installed in the Servo Drives New Generation as a module (option F9). Pin assignment and signal levels comply with Profinet standard IEC 61918 or IEC 61784-5. The bus connection is galvanically isolated from the servo drive.

Further characteristics of the Profinet interface of servo drives:

Software	Stack software in binary or source format
Requirements	100 Mbit/s full duplex, switched Ethernet
Language for GSD file	GSDML
Topologies	star, line, tree, and ring
Functionality	• PROFINET device according to specification V2.3, conformance class B • Media redundancy client • Multicast provider and subscriber
Number of PROFINET controllers for simultaneous communication (shared devices)	2
Number of connections per PROFINET controller	2
Protocols	Ethernet, IP, ARP, DCP, DHCP, RPC, RT, IRT, ESR
Max. size I/O data per communication relationship	1,024 bytes
Data exchange	• I/O data cyclic • Alarms, read and write access to records also acyclic
Max. number of participants	unlimited
Max. configuration data in the device	8 kbyte
Max. parameter data in the device	8 kbyte
Max. I/O data per communication relationship	1,440 bytes
Support of profiles	yes

4 Profinet Introduction

PROFINET (Process Field Network) is the open Industrial Ethernet standard by Profibus & Profinet International (PI) for automation. Profinet uses TCP/IP and IT standards, is compatible with real-time Ethernet and permits the integration of field bus systems.

Profinet has a modular structure so that the functionality can be selected by the user. It mainly differs in the way data are exchanged in order to meet the speed requirements.

For Profinet, variants Profinet IO and Profinet CBA exist:

- Profinet IO (Input - Output) was created for the connection of local periphery to a controller. For the various fields of application, the available functions and real-time properties are divided into the three conformity classes CC-A, CC-B, and CC-C.
- Profinet CBA (Component Based Automation) is intended for component-based communication via TCP/IP and real-time communication for real-time requirements in modular plant engineering. Both communication types can be used in parallel.

Profinet IO and Profinet CBA can communicate simultaneously in the same bus system. They can be operated separately as well as combined so that a facility part with Profinet IO appears like a Profinet CBA facility from the point of view of the facility.

For the Servo Drives New Generation, Profinet IO is used.

A Profinet IO system consists of the following devices:

- IO controller
- IO device (field device). It consists of several modules and sub-modules. The sub-modules include the individual input and output signals for the process.
- IO supervisor is a development tool, typically based on a PC in order to parameterize and diagnose the individual IO devices.

A minimum Profinet IO system consists of at least one IO controller controlling one or several IO devices. Additionally, one or several IO supervisors can be added temporarily, if required.

When two IO systems exist in the same IP network, the IO controller can also share an input signal as shared-input by accessing the same sub-module in an IO device in read mode.

Each automation device with an Ethernet interface can fulfill the functionality of an IO controller as well as an IO device at the same time.

To increase the availability, Profinet can also be implemented with a system redundancy. In this case, two IO controllers controlling the same IO devices are configured.

An application relation (AR) is built up between an IO controller and an IO device. Via this AR, communication relations (CR) with different characteristics are defined:

- Record Data CR for acyclic parameter transfer
- IO Data CR for cyclic process data exchange
- Alarm CR for signaling alarms in real-time

5 Terms and Abbreviations

This section provides a brief overview of the major terms and abbreviations. It does not replace the original documents or proper training.

Acyclic parameter data (Record Data CR)

Acyclic data traffic is used for not continually recurring events. Examples of acyclic data traffic are the sending of parameterization and configuration data during the start of a peripheral device to the device or the sending of a diagnose message from the peripheral device to the central unit in running operation.

Acyclic data use the UDP/IP protocol.

Alarms (acyclic alarm data)

Alarms are special acyclic messages transmitted by the periphery device to the controller, if required. These are time-critical and thus transmitted directly via Ethernet like the cyclic data. However, in contrast to the cyclic data, they have to be acknowledged by the recipient.

Bus name

Unique device name of a communication participant in the Profinet network.

Cyclic data (IO Data CR)

Contents of the cyclic data traffic are data the central unit sends to the periphery as well as data a peripheral device reads at its inputs and sends to the central unit for processing. Usually, in each cycle such a cyclic data package is sent from the central unit to the peripheral device and the other way round.

Extended services

Transmission method in parameter communication; permits transmission of parameters > 4 bytes.

See “Parameter channel”.

GSDML file

Electronic data sheet according to EN 50170; text file in ASCII format.

With GSDML files, a Profinet system can be configured openly (independent of the manufacturer). One or more drives with Profinet interface are introduced to the Profinet master using the GSDML file as input for a configuration tool.

Input data

Process data the slave uses to respond to the master.

IO Data CR

See cyclic data.

IP address

The IP address of the servo drive in the Profinet network is usually defined in the Profinet configuration tool and transmitted to the servo drive when the bus runs up. Alternatively, it can be stored in the servo drive.

Master/slave application

The master or slave applications are often called applications. From the point of view of Profinet, an application is the software installed on top of the Profinet software.

• Master application:	e. g. PLC program
• Slave application:	firmware

Modules, module descriptions

Profinet interfaces can also be run as modules. One or more module descriptions are stored between the key words *Module* and *EndModule* in the GSDML file (defined module types). Up to Max_Modules can be activated from these module types during configuration.

Several modules are defined in the GSDML file of the servo drives. Each module can be activated only once, but does not have to be activated.

Output data

Process data the master sends to the slave.

Parameters

(Generally) a class of data transmitted by individual access. Parameter communication requires that the parameters are addressed using index and subindex.

The servo drive parameters and their addresses (indexes) are listed in operating instructions 6710.201 “Functions and Parameters”.

Parameter channel

The part of cyclic communication that provides for access to the servo drive parameters (objects) using Profinet.

When the parameter channel is active, a master application gets access to all parameters of the servo drives (in addition to or instead of the process data control).

Two methods are available for data transmission:

- With standard services, parameters up to 4 bytes long can be transmitted in a request/response pair.
- With extended services, parameters of more than 4 byes can be transmitted.

Segments of data are transmitted in a sequence of request/response pairs (segmented transfer).

Process data

(Generally) a class of data required for controlling and monitoring a process.

As per Profinet, process data is exchanged as user data between the master and one or more slaves in recurring cycles.

Direction information always refers to the master. The master sends its cyclic output data to the slave, and the slave responds by sending its cyclic input data.

“Cyclic” is actually redundant with regards to Profinet (and therefore input and output data) because process data always refers to data exchanged in a cycle.

Process data channel

The part of cyclic communication that provides for continual cyclic access to the servo drive process data via Profinet.

Record Data CR

See acyclic parameter data.

Segmented transfer

Method of extended services in parameter communication; long parameters are divided into segments transmitted in a sequence of request/response pairs.

See Parameter channel.

Standard Services

Method of transmission in parameter communication; permits transmission of parameters of up to 4 bytes in length.

See Parameter channel.

6 Connection and Commissioning

For connection and status display, the servo drives are equipped with the following elements:

- bus connection
- LEDs
- coding switch

These elements are located at the front panel of the servo drives with Profinet interface.

6.1 Bus Connection

X4.1/F7 Profinet-In: RJ45 connector

X4.2/F7 Profinet-Out: RJ45 connector

The Profinet interface consists of a Profinet switch with 2 connectors. Connector assignment and signal levels comply with Ethernet standard IEEE 802.3 as well as standard IEC 61918 or 61784-5.

Connectors X 4.1 and X 4.2 both of which can be used for this connection are located at the front panel of the servo drives with Profinet interface. Only one of these ports is required for the connection with the Profinet network, the second port is intended for the optional connection of further devices.

Due to the wiring with standard Ethernet, a termination of the first or last bus participant is not required.

6.2 Coding Switch

The coding switch has 16 different positions (0-F). The first 15 positions (0-E) set pre-defined device names, in position F, the name must be assigned via the configuration tool or the name assigned and stored there is used.

Switch Position	Device Name
0	ac-servo-ng
1	ac-servo-ng-1
2	ac-servo-ng-2
...	
D	ac-servo-ng-13
E	ac-servo-ng-14
F	configuration tool

6.3 LEDs

The bus LEDs of the servo drives are used for displaying internal states of the Profinet stack. Some internal states are output via different flash codes.

LED Error (red)

Status	Flash Sequence	Description
NO-ERROR	Off	No error
DOUBLE_ADDRESS_ERROR	Flashing 2×, then 1 s off	Address conflict detected. Network communication might be distur-bed.
HARDWARE_ERROR	Flashing 3×, then 1 s off	Possible causes are incorrect ac-cess to the MAC or interrupted da-ta flow.
APPL_WATCHDOG_EXPIRED	Flashing 1×shortly, then 1 s off	Application watchdog expired.
PROTOCOL_ERROR	On	A protocol-specific error has occur-red.

LED Run (green)

Status	Flash Sequence	Description
BEFORE_INIT	Off	Stack not initialized
INIT	Flashing 1×, then 1 s off	Stack being initialized
ONLINE	Blinking	Stack is online (device can be ac-cessed via the network interface)
CONNECTED	On	Stack is online (device can be ac-cessed via the network interface)

LED Aux1 (yellow)

Status	Flash Sequence	Description
NO_PAR	Off	Parameter channel (PZD8) not configured or configured but not used yet.
PAR_ACCESS	Blinking	Parameter channel (PZD8) active, communication takes place
PAR_USED	On	Parameter channel (PZD8) active, however no communication at the moment.

LED Aux2 (yellow)

LED Aux2 is reserved for the flash function and therefore only active after a corresponding function triggering by the Profinet configuration tool.

7 Activating and Deactivating Modules

Like with Profibus, with Profinet, as well, modules for process data communication are pre-defined. The modules are plugged onto the so-called module slots. Currently, the servo drive is equipped with 64 module slots only the first 6 of which can be used as only 6 different modules are defined in the GSDML and these are assigned to defined module slots. Currently, the the following modules are defined in the GSDML file:

Module	**Slot**	**Function**
PAR8 IN/OUT	1	Parameter channel
PZD16 IN/OUT	2	Process data channel
PAR8 IN	3	Alternative parameter channel IN
PAR8 OUT	4	Alternative parameter channel OUT
PZD16 IN	5	Test channel IN
PZD16 OUT	6	Test channel OUT

PZD = Process data channel; PAR = Parameter channel

As standard, only the first two modules are assigned. Assignment and function of the PAR8 IN/OUT modules correspond to the Profibus PAR module (parameter channel), PZD16 IN/OUT corresponds to the Profibus PZD module (process data channel). A combined 24 byte PAR+PZD module does not exist. It is not required as the Profinet modules can be used independently of each other. Via modules PAR8 IN and PAR8 OUT, a second parameter channel can be operated, modules PZD16 IN und PZD16 OUT are for internal test purposes, only, and should not be used.

7.1 PAR8 Module

Function and assignment of the PAR8 module correspond exactly to the Profibus PAR module, all services defined there are supported. The PAR module is defined as octet string 8, i. e as 8-byte array without further function identification of the individual bytes.

7.2 PZD16 Module

Function and assignment of the PZD16-Modul correspond to the Profibus PZD module. In the GSDML file, the PZD16 channel is defined as structure with its actual assignment (control word, target position, status word, actual position ...). Depending on the configuration tool, this information is not used, like with Profibus, PZD16 is treated as 16-byte array. Other tools interpret this information correctly and provide the individual variables of the process data channel.

7.3 Module Parameters

Other parameters accessible via the configuration tool are assigned to the PAR IN/OUT and PZD IN/OUT standard modules. These are:

Parameter	Default	Function
Activate diagnose alarm	Off	Activates the report of alarms via this module.
Activate connection monitoring	Off	Activates the connection monitoring via this module.

7.4 Connection Monitoring

If the connection monitoring of at least one module is activated, a fault is triggered in the servo drive in case of a connection loss after the bus run-up. Thus, this function roughly corresponds to the master watchdog monitoring of Profibus.

8 Process Data Communication

The process data communication of the Profinet interface of the servo drives is realized via a so-called process data channel.

The process data channel contains 16 bytes per direction (output and input process data). On the process data channel, variables of the digital servo drives are mapped; the mapping of the variables on the individual bytes of the process data channel is described in the following tables, one each for the output and the input process data.

If the process data channel is active, the length of the cyclic data will increase by 16 bytes. The process data channel occupies the last 16 bytes in the cyclic channel.

Other process data channel lengths and mapping of other variables are possible on request, please contact ESR.

The function of the variables is described in operating instructions 6710.201 "Functions and Parameters". Please, pay particular attention to section Variable Descriptions.

? For process data communication in the cyclic channel, the correct user addresses in the master must be assigned to the individual input and output data of the cyclic communication.

8.1 Output Process Data

The output process data are transmitted from the master (e. g. PLC, PC) to the servo drive.

Profinet Output Process Data		
Process Data		**Variable**
Byte	**Bit**	
1		Axis control word (bit 0 .. 7)
2		Axis control word (bit 8 .. 15)
3		Target velocity (bit 31 .. 24)
4		Target velocity (bit 23 .. 16)
5		Target velocity (bit 15 .. 8)
6		Target velocity (bit 7 .. 0)
7		Target position (bit 31 .. 24)
8		Target position (bit 23 .. 16)
9		Target position (bit 15 .. 8)
10		Target position (bit 7 .. 0)
11	7 .. 4	Digital outputs O 2.3 .. 2.0 (software outputs)
	3 .. 0	Digital outputs O 1.3 .. 1.0 (switching outputs at X7)
12		Digital inputs I 9.7 .. 9.0 (software inputs)
13		Torque setpoint (bit 15 .. 8)
14		Torque setpoint (bit 7 .. 0)

Profinet Output Process Data		
Process Data		Variable
Byte	Bit	
15		Axis operating mode (bit 15 .. 8)
16		Axis operating mode (bit 7 .. 0)

8.2 Input Process Data

The input process data are transmitted from the servo drive to the master (e. g. PLC, PC).

Profinet Input Process Data		
Process Data		Variable
Byte	Bit	
1		Axis status word (bit 0 .. 7)
2		Axis status word (bit 8 .. 15)
3		Actual velocity (bit 31 .. 24)
4		Actual velocity (bit 23 .. 16)
5		Actual velocity (bit 15 .. 8)
6		Actual velocity (bit 7 .. 0)
7		Actual position (bit 31 .. 24)
8		Actual position (bit 23 .. 16)
9		Actual position (bit 15 .. 8)
10		Actual position (bit 7 .. 0)
11		Digital inputs I 1.7 .. 1.0 (switching inputs at X7)
12		Digital inputs I 3.7 .. 3.0
13		Actual current (bit 15 .. 8)
14		Actual current (bit 7 .. 0)
15		Digital outputs O 8.7 .. 8.0 (software outputs)
16		Digital outputs O 9.7 .. 9.0 (software outputs)

9 Parameter Communication

With the Profinet interface, parameter communication is realized via a parameter channel. With the parameter channel, a Profinet master can access all servo drive parameters (variables). Variables can be read and written via the parameter channel and are addressed using index and subindex. Operating instructions 6710.201 “Functions and Parameters” contain more specific information on variables.

The parameters are transmitted in the cyclic channel in addition to the usual process data. The parameter channel can be activated or deactivated by the master.

When the parameter channel is active, the length of the cyclic data increases by 8 bytes. The parameter channel occupies the first 8 bytes in the cyclic channel.

9.1 Standard and Extended Services

All parameters (variables) of the servo drive up to a length of 4 bytes can be accessed with standard services. Parameters longer than 4 bytes can be transmitted with extended services, as well. This only applies to a few parameters that do not need to be accessed by the master in most applications:

Parameters > 4 byte			
Index	**Name**	**Data Type**	**Comment**
5ee6	Motor description	VisStr 16	
5f43	Trace buffer 1	Array 16 of OctStr 128	Buffer for data recording; used by the oscilloscope functions of the SPP Windows command and commissioning software
5f45	Trace buffer 2		
5f47	Trace buffer 3		
5f5f	Part program	Array 71 of OctStr 128	Created with the SPP Windows part program editor
5f81	Fault detail	Array 4 of VisStr 16	Additional information for a few axis error codes

The standard services or extended services transmission method is selected using the service byte.

9.2 Standard Services

Almost all servo drive parameters (variables) can be accessed with standard services.

If the master application does not require access to the few parameters that are longer than 4 bytes, only standard services have to be implemented. This reduces the implementation efforts required on the master side considerably.

9.2.1 Parameter Channel Structure

The parameter channel structure is identical for both transmission directions, as a request in the output data from the master to the slave and as a response in the input data from the slave to the master.

The master must not send a second request until it receives the response to the first request.

When the parameter channel is active, in standard services, the following 8 bytes are transmitted in the cyclic channel prior to the actual process data:

Parameter Channel, Standard Services	
Byte	**Contents**
1	Service byte
2	Subindex
3	Index (bit 15 .. 8)
4	Index (bit 7 .. 0)
5	Data/Error 1 (highest-value byte)
6	Data/Error 2
7	Data/Error 3
8	Data/Error 4 (lowest-value byte)

The request and response control of the parameter channel is carried out via the service byte (byte 1). The service byte also selects standard services as transmission method. The following section describes the service byte structure in detail.

Bytes 2 to 4 address the parameter. The request contains indexes and subindexes for addressing the parameter. The index and subindex actually used to address the parameter are returned in the response.

Bytes 5 to 8 contain data or an error code.

9.2.2 Service Byte Structure

The service byte (byte 1 in the parameter channel) is the first byte transmitted. It has the lowest relative address in the cyclic channel (address 0).

The individual bits have the following meaning:

Service Byte (Standard Services)	
Bit	**Contents**
0 .. 2	Service coding: • 000 $_{bin}$ = 0: no request, standard services • 001 $_{bin}$ = 1: read request, standard services (read data from servo drive) • 010 $_{bin}$ = 2: write request, standard services (write data to servo drive) • 100 $_{bin}$ = 4: extended services (read or write)
3	Reserved (redundant, however, should always be set to 0)
4 .. 5	Standard services: length of data (number of valid byes) in the Data/Error fields • 00 $_{bin}$ = 0: 1 byte • 01 $_{bin}$ = 1: 2 byte • 10 $_{bin}$ = 2: 3 byte • 11 $_{bin}$ = 3: 4 byte Extended services: code for read or write request
6	Handshake (code that initiates a new request): The master changes this bit for every new request. The servo drive copies this bit in its response message.
7	Status (error information from servo drive): With this bit, the servo drive response informs the master on whether or not the request was completed successfully. • 0 = request completed successfully • 1 = request not completed; fault occurred. The data in the Data/Error field is an error message.

The required service is coded in bits 0 to 2. The values 001 $_{bin}$ (read request) and 010 $_{bin}$ (write request) are reserved for the master since a slave must not send requests. ESR servo drives respond with 000 $_{bin}$ (no request).

When the ESR servo drives receive an invalid service coding, they respond with an error status message (error code 06 05 00 10 $_{hex}$, Invalid Request Parameter).

In standard services, bits 4 and 5 specify the data length (number of valid bytes) in Data/Error fields 1 to 4:

- For write requests from the master and read responses from the slave, the number of bytes transmitted in the Data/Error fields 1 to 4 must be specified (00 $_{bin}$ for 1 byte, 01 $_{bin}$ for 2 bytes, etc.). The receiver checks this information.
- For read requests from the master and write responses from the slave, this information is redundant and not necessary. The receiver does not need to check the information. ESR servo drives always send 11 $_{bin}$ in a write response as well as in an error status response because the length of error codes is always 4 bytes.

The parameter channel is controlled with bit 6 (handshake). The request and response messages are transmitted in the cyclic channel until they are overwritten by the users again. There are two different states at the bus:

- Request and response handshake bits do not match:

  - The master sent a new request and is waiting for a response. The master's request message is valid.

    The master checks the handshake bits in the responses from the slave and does not carry out any additional action. It must not send any further requests until it receives a response to the current request from the slave (with the same handshake bit).

  - The slave detects the new request and processes it. The slave's old response message is invalid.

    Once the slave receives the new request, it no longer checks the handshake bits in the requests from the master. The slave copies the handshake bit in the response to the new request (the handshake bits in request and response now match).

- Request and response handshake bits match:

  - The slave sent a response and is waiting for a new request. The slave's response message is valid.

    The slave checks the handshake bits in the requests from the master and does not carry out any additional action. It will not send any further responses until it receives a new request from the master (with a different handshake bit).

  - The master detects the slave's response and processes it. The master's old request message is invalid.

    Once the master receives the response, it no longer checks the handshake bits in the responses from the slave. When it creates a new request, it changes the handshake bit (the handshake bits in the request and response do not match).

The following procedure is recommended for the master side:

- The handshake bit is immediately set to 0 following initialization.
- The master reads the handshake bit of the last request message (still on the bus), inverts it and transmits it in its new request message.
- The service byte (byte 1) with the changed handshake bit is the last transmitted through the bus. The new request is then valid.

Bit 7 (status) specifies the contents of the Data/Error fields in the response from the slave:

- Status = 0: data

  A parameter value that occupies 1 to 4 bytes depending on the data format is transmitted to the Data/Error fields.

- Status = 1: error

  The Data/Error fields contain information describing an error.

This bit must be set to 0 in the master's request.

9.2.3 Transmission Format

Parameter values up to 4 bytes in length can be transmitted in the Data/Error fields. If the parameter is longer than 1 byte, it is transmitted in Motorola format (highest-value byte/word first). Parameters shorter than 4 bytes are transmitted in the first data fields:

Parameter Channel		Transmitted Element			
Byte	**Data**	**Byte**	**Word**	**3 Bytes ***	**Double Word**
5	Data 1	Bit 7 .. 0	Bit 15 .. 8	Byte 1	Bit 31 .. 24
6	Data 2	–	Bit 7 .. 0	Byte 2	Bit 23 .. 16
7	Data 3	–	–	Byte 3	Bit 15 .. 8
8	Data 4	–	–	–	Bit 7 .. 0

* length 3 character strings only

9.2.4 Examples: Parameter Communication with the Servo Drive

For parameter communication in the cyclic channel, input and output data of cyclic communication must be assigned to the correct user data addresses in the Profinet master.

The request should be built up “backwards” so that incomplete requests cannot be sent to the servo drive and processed by it. The first (service) byte, which, along with the handshake bit, signals a new request to the servo drive, is set last.

The master should wait for the slave’s response only for a limited period of time. The implementation of a time monitoring feature is useful.

A master’s read or write request is created as follows (in this sequence):

- Data/Error field:
  - read request: enter 0.
  - write request: enter the parameter value.
- Enter the address of the required parameter in the Index and Subindex fields.
- Service byte:
  - Set the service code bits according to the required request type (read or write).
  - Enter the length of the parameter in the length bit.
  - Set the status bit to 0 (no error).
  - Change the handshake bit.
- Start the time monitoring feature.

Follow these steps to evaluate the servo drive’s response:

- Check whether or not the handshake bit in the input data is identical with the one in the output data.

  If it is, the response was received and the time monitoring feature can be stopped. If time runs out and no new response has been received, the master should send a corresponding error message.

- Check whether or not the status bit is set:
  - If the bit is not set, the request was carried out successfully. The Data/Error field contains the required parameter value (read response) or is empty (write response).
  - If the bit is set, the request was not carried out. An error occurred. The Data/Error field contains information on the error.

9.2.4.1 Writing Parameters (Example)

The max current amount (index 6073 $_{hex}$, data type unsigned16 = word) is to be set to 200%.

- Message from the master to the servo drive:

  5200 6073 07D0 0000 $_{hex}$ (if handshake bit was previously 0)
  1200 6073 07D0 0000 $_{hex}$ (if handshake bit was previously 1)

Byte	Value	Contents	
1	52 $_{hex}$ or 12 $_{hex}$	Bit 0 .. 2:	xxxx x010 = write request
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx01 xxxx = 2 byte data in Data/Error field
		Bit 6:	x1xx xxxx – if handshake was previously 0
			x0xx xxxx – if handshake was previously 1
		Bit 7:	0xxx xxxx = no error
2	00 $_{hex}$	Subindex = 00 $_{hex}$	
3	60 $_{hex}$	Index = 6073 $_{hex}$	
4	73 $_{hex}$		
5	07 $_{hex}$	Data =07D0 $_{hex}$ (= 2000)	
6	D0 $_{hex}$		
7	00 $_{hex}$		
8	00 $_{hex}$		

- Drive response if carried out without errors:

  5000 6073 0000 0000 $_{hex}$ (if handshake bit was 0)
  1000 6073 0000 0000 $_{hex}$ (if handshake bit was 1)

Byte	Value	Contents	
1	50 $_{hex}$ or 10 $_{hex}$	Bit 0 .. 2:	xxxx x000 = no request
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx11 xx01 = 2 byte data in Data/Error field
		Bit 6:	x1xx xxxx = sends back handshake 1
			x0xx xxxx = sends back handshake 0
		Bit 7:	0xxx xxxx = request carried out
2	00 $_{hex}$	Subindex = 00 $_{hex}$	
3	60 $_{hex}$	Index = 6073 $_{hex}$	
4	73 $_{hex}$		
5 .. 8	00 $_{hex}$	Data = 0000 0000 $_{hex}$ (no meaning)	

9.2.4.2 Reading Parameters (Example)

The DC bus voltage (index 6079 $_{hex}$, data type unsigned16 = word) is to be read out.

- Message from the master to the servo drive:

  5100 6079 0000 0000 $_{hex}$ (if handshake bit was previously 0)
  1100 6079 0000 0000 $_{hex}$ (if handshake bit was previously 1)

Byte	Value	Contents	
1	51 $_{hex}$ or 11 $_{hex}$	Bit 0 .. 2:	xxxx x001 = read request
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx01 xxxx = 2 byte data in Data/Error field
		Bit 6:	x1xx xxxx – if handshake was previously 0
			x0xx xxxx – if handshake was previously 1
		Bit 7:	0xxx xxxx = no error
2	00 $_{hex}$	Subindex = 00 $_{hex}$	
3	60 $_{hex}$	Index = 6079 $_{hex}$	
4	79 $_{hex}$		
5 .. 8	00 $_{hex}$	Data = 0000 0000 $_{hex}$ (no meaning)	

- Drive response if carried out without errors:

  5000 6079 0140 0000 $_{hex}$ (if handshake bit was 1)
  1000 6079 0140 0000 $_{hex}$ (if handshake bit was 0)

Byte	Value	Contents	
1	50 $_{hex}$ or 10 $_{hex}$	Bit 0 .. 2:	xxxx x000 = no request
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx01 xxxx = 2 byte data in Data/Error field
		Bit 6:	x1xx xxxx = sends back handshake 1
			x0xx xxxx = sends back handshake 0
		Bit 7:	0xxx xxxx = request carried out
2	00 $_{hex}$	Subindex = 00 $_{hex}$	
3	60 $_{hex}$	Index = 6079 $_{hex}$	
4	79 $_{hex}$		
5	01 $_{hex}$	Data = 0140 $_{hex}$ (= 320 V)	
6	40 $_{hex}$		
7	00 $_{hex}$	Data = 0000 $_{hex}$ (no meaning)	
8	00 $_{hex}$		

9.2.4.3 Response with Error Status (Example)

The motor description (*MotorName*, index 5EE6 $_{hex}$, data type visiblestring16 = 16-bit long character string) is tried to be read using a standard read request (up to 4 bytes in length).

- Message from the master to the servo drive:

  7100 5EE6 0000 0000 $_{hex}$ (if handshake bit was previously 0)
  3100 5EE6 0000 0000 $_{hex}$ (if handshake bit was previously 1)

Byte	Value	Contents	
1	71 $_{hex}$ or 31 $_{hex}$	Bit 0 .. 2:	xxxx x001 = read request
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx11 xxxx = 4 byte data in Data/Error field
		Bit 6:	x1xx xxxx – if handshake was previously 0
			x0xx xxxx – if handshake was previously 1
		Bit 7:	0xxx xxxx = no error
2	00 $_{hex}$	Subindex = 00 $_{hex}$	
3	5E $_{hex}$	Index = 5EE6 $_{hex}$	
4	E6 $_{hex}$		
5 .. 8	00 $_{hex}$	Data = 0000 0000 $_{hex}$ (no meaning)	

- Drive response with error status:

  F000 5EE6 0605 0012 $_{hex}$ (if handshake bit was 1)
  B000 5EE6 0605 0012 $_{hex}$ (if handshake bit was 0)

Byte	Value	Contents	
1	F0 $_{hex}$ or B0 $_{hex}$	Bit 0 .. 2:	xxxx x000 = no request
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx11 xxxx = 4 byte data in Data/Error field
		Bit 6:	x1xx xxxx = sends back handshake 1
			x0xx xxxx = sends back handshake 0
		Bit 7:	1xxx xxxx = request not carried out, error
2	00 $_{hex}$	Subindex = 00 $_{hex}$	
3	5E $_{hex}$	Index = 5EE6 $_{hex}$	
4	E6 $_{hex}$		
5	06 $_{hex}$	Error code = 06 05 00 12 $_{hex}$ (data is too long or too short)	
6	05 $_{hex}$		
7	00 $_{hex}$		
8	12 $_{hex}$		

9.3 Extended Services

Extended services is used to transmit parameters longer than 4 bytes. It is allowed to transmit parameters shorter than or equal to 4 bytes using extended services (however, 2 request/response pairs are required for that).

If the master application does not require access to the few parameters longer than 4 bytes, only standard services must be implemented. In this case, the implementation effort for extended services required on the master side can be reduced considerably.

9.3.1 Parameter Channel Structure

The parameter channel structure is identical for both transmission directions, as a request in the output data from the master to the slave and as a response in the input data from the slave to the master.

The master must not send a second request until it receives the response to the first request.

When the parameter channel is active, extended services transmits the following 8 bytes in the cyclic channel prior to the actual process data:

Parameter Channel, Extended Services		
Byte	**Contents of first message (initiate) or error status message**	**Contents of following messages (read segment or write segment)**
1	Service byte	Service byte
2	Subindex	Control byte
3	Index (bit 15 .. 8)	Data 1
4	Index (bit 7 .. 0)	Data 2
5	Data/Error 1 (highest-value byte)	Data 3
6	Data/Error 2	Data 4
7	Data/Error 3	Data 5
8	Data/Error 4 (lowest-value byte)	Data 6

The structure of first message and error status message is identical with the structure in standard services. Additional bytes are used to transmit data in the following messages. The control byte controls sequential transmission.

Control of requests and responses in the parameter channel is carried out via the service byte (byte 1). It is also used for selecting extended services as transmission method.

The control byte specifies the number of valid data bytes in the fields Data 1 to Data 6. It contains a status bit that indicates whether additional segments follow or the transmission of the parameter is terminated.

9.3.2 Service Byte Structure

The service byte (byte 1 in the parameter channel) is the first byte transmitted. It has the lowest relative address in the cyclic channel (address 0).

The individual bits have the following meaning:

Service Byte (Extended Services)	
Bit	**Contents**
0 .. 2	Service coding: • 000 $_{bin}$ = 0: data transmission aborted by drive or response to abort by master (bit 7 = 1), extended services • 100 $_{bin}$ = 4: extended services (read or write, see bits 4 and 5) • 001 $_{bin}$ = 1 and 010 $_{bin}$ = 2: read or write request, standard services
3	reserved (redundant, however, should always be set to 0)
4 .. 5	Extended services: codes for read or write request • 00 $_{bin}$ = 0: initiate read request (initiate segment read) or no request (in slave’s response) • 01 $_{bin}$ = 1: read data segment from servo drive (read segment) • 10 $_{bin}$ = 2: initiate write request (initiate segment write) • 11 $_{bin}$ = 3: write data segment to servo drive (segment write) Standard services: length of data
6	Handshake (code that initiates a new request): The master changes this bit for every new request. The servo drive copies the bit in its response message.

Service Byte (Extended Services)	
Bit	**Contents**
7	Status (error information from servo drive): The servo drive uses this bit in its response to inform the master whether or no the request was carried out successfully. In extended services, value 1 causes an abort of the sequential data transmission. The master may also initiate this abort. • 0 = request carried out successfully • 1 = request not carried out; error occurred. Transmission of sequential data stops. Data in Data/Error field is an error message.

Bits 0 to 2 are coded with 100 $_{bin}$ in extended services. While in standard services, the slave always responds with service code 000 $_{bin}$ (= no request), in extended services it responds with service code 100 $_{bin}$ (= extended services). Thus, a clear distinction can be made between the responses in extended services and the responses in standard services.

Upon receipt of an invalid service code, ESR servo drives respond with an error status message (error code 06 05 00 10 $_{hex}$, invalid request parameter).

Bits 4 and 5 specify the type of request in extended services (read or write).

As in standard services, the parameter channel is controlled with bit 6 (handshake).

Bit 7 (status) specifies the contents of the Data/Error fields in the slave's response. When status = 1, sequential transmission is canceled. The master can also initiate this abort.

9.3.3 Control Byte Structure

The control byte is transmitted as byte 2 in the parameter channel in sequential data transmission from the second message onward.

The individual bits have the following meaning:

Control Byte	
Bit	**Contents**
0	Sequential transmission status: • 0: additional segments exist • 1: no additional segments exist
1 .. 3	Number of valid data bytes in Data 1 .. Data 6, valid values: 001 $_{bin}$ (1) to 110 $_{bin}$ (6)
4 .. 7	reserved (redundant, however, should always be set to 0)

9.3.4 Sequential Data Transmission

9.3.4.1 Initiate Read Request (Initiate Segment Read)

The master specifies the transmission method (bits 0 to 2 in service byte = 100 $_{bin}$, extended services) and the type of request (bits 4 and 5 = 00 $_{bin}$, Initiate Segment Read) with the service byte in the initial read request message to the drive. The address of the parameter to be read is specified in the subindex and index fields.

Message to Drive System: Initiate Segment Read in Extended Services			
Byte	**Value**	**Contents**	
1	44 $_{hex}$ or 04 $_{hex}$	Bit 0 .. 2:	xxxx x100 = extended services
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx00 xxxx = starts a new read request (Initiate Segment Read)
		Bit 6:	x1xx xxxx – if handshake was previously 0
			x0xx xxxx – if handshake was previously 1
		Bit 7:	0xxx xxxx = no error
2	...	Subindex	
3 .. 4	...	Index	
5 .. 8	00 $_{hex}$	Data = 0000 0000 $_{hex}$ (no meaning)	

If the request was carried out successfully, the drive returns the length of the parameter as a 32-bit value without prefix (in Motorola format) in its response:

Response from Drive System (to Initiate Segment Read)			
Byte	**Value**	**Contents**	
1	44 $_{hex}$ or 04 $_{hex}$	Bit 0 .. 2:	xxxx x100 = extended services
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx00 xxxx = no request
		Bit 6:	x1xx xxxx – if handshake was previously 0
			x0xx xxxx – if handshake was previously 1
		Bit 7:	0xxx xxxx = no error
2	...	Subindex	
3 .. 4	...	Index	
5	...	Length (bit 31 .. 24)	
6	...	Length (bit 23 .. 16)	
7	...	Length (bit 15 .. 8)	
8	...	Length (bit 7 .. 0)	

The current maximum length of a parameter is 128 bytes.

9.3.4.2 Read Segment

Once the read request is initiated, one or more read segment requests (bits 4 and 5 in service byte = 01 $_{bin}$, read segment) follow according to the data length:

Message to Drive System: Read Segment			
Byte	**Value**	**Contents**	
1	54 $_{hex}$ or 14 $_{hex}$	Bit 0 .. 2:	xxxx x100 = extended services
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx01 xxxx = read segment
		Bit 6:	x1xx xxxx – if handshake was previously 0
			x0xx xxxx – if handshake was previously 1
		Bit 7:	0xxx xxxx = no error
2	00 $_{hex}$	Control byte = 00 $_{hex}$ (no meaning)	
3 .. 8	00 $_{hex}$	Data = 0000 0000 0000 $_{hex}$ (no meaning)	

The slave uses the control byte to specify the number of valid data bytes in Data 1 .. Data 6 (nnn = 1 to 6, beginning with Data 1) and indicates whether or not additional segments will follow:

Response from Drive System (to read segment)			
Byte	**Value**	**Contents**	
1	44 $_{hex}$ or 04 $_{hex}$	Bit 0 .. 2:	xxxx x100 = extended services
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx00 xxxx = no request
		Bit 6:	x1xx xxxx – if handshake was previously 0
			x0xx xxxx – if handshake was previously 1
		Bit 7:	0xxx xxxx = no error
2	0x $_{hex}$	Bit 0 :	xxxx xxx0 – if additional segments follow
			xxxx xxx1 – if no additional segments follow
		Bit 1 .. 3:	xxxx nnnx = number of valid data bytes in Data 1 to Data 6
		Bit 4 .. 7:	0000 xxxx (reserved)
3	...	Data 1	
4	...	Data 2	
5	...	Data 3	
6	...	Data 4	
7	...	Data 5	
8	...	Data 6	

If less than 6 data bytes are transmitted, the valid data is always at the beginning (for example, nnn = 4: Data 1 to Data 4 valid).

9.3.4.3 Initiate Write Request (Initiate Segment Write)

The master uses this service byte when initiating a write request message to the drive to specify transmission method (bits 0 to 2 in service byte = 100 $_{bin}$, extended services) and request type (bits 4 and 5 = 10 $_{bin}$, start a write request, Initiate Segment Write). The address of the parameter to be written is specified in the subindex and index fields. The length of the parameter is specified as a 32-bit value without a prefix (in Motorola format) in the Data 1 to Data 4 fields.

Message to Drive System: Initiate Segment Write in Extended Services			
Byte	**Value**	**Contents**	
1	64 $_{hex}$ or 24 $_{hex}$	Bit 0 .. 2:	xxxx x100 = extended services
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx10 xxxx = start a write request (initiate segment write)
		Bit 6:	x1xx xxxx – if handshake was previously 0
			x0xx xxxx – if handshake was previously 1
		Bit 7:	0xxx xxxx = no error
2	...	Subindex	
3 .. 4	...	Index	
5	...	Length (bit 31 .. 24)	
6	...	Length (bit 23 .. 16)	
7	...	Length (bit 15 .. 8)	
8	...	Length (bit 7 .. 0)	

The current maximum length of a parameter is 128 bytes.

If the request was carried out successfully, the drive system confirms the request in its response:

Response from Drive System (to Initiate Segment Write)			
Byte	**Value**	**Contents**	
1	44 $_{hex}$ or 04 $_{hex}$	Bit 0 .. 2:	xxxx x100 = extended services
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx00 xxxx = no request
		Bit 6:	x1xx xxxx – if handshake was previously 0
			x0xx xxxx – if handshake was previously 1
		Bit 7:	0xxx xxxx = no error
2	...	Subindex	
3 .. 4	...	Index	
5 .. 8	00 $_{hex}$	Data = 0000 0000 $_{hex}$ (no meaning)	

9.3.4.4 Write Segment

After the write request has been initiated, one or more write data segment requests (bits 4 and 5 in service byte = 11 $_{bin}$, write segment) follow depending on the length of the data. The master uses the control byte to specify the number of valid data bytes in Data 1 .. Data 6 (nnn = 1 to 6, beginning with Data 1) and indicates whether or not additional segments will follow:

Message to Drive System: Write Segment

Byte	Value	Contents	
1	54_{hex} or 14_{hex}	Bit 0 .. 2:	xxxx x100 = extended services
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx11 xxxx = writes data segment
		Bit 6:	x1xx xxxx – if handshake was previously 0
			x0xx xxxx – if handshake was previously 1
		Bit 7:	0xxx xxxx = no error
2	$0x_{hex}$	Bit 0 :	xxxx xxx0 – if additional segments follow
			xxxx xxx1 – if no additional segments follow
		Bit 1 .. 3:	xxxx nnnx = number of valid data bytes in Data 1 to Data 6
		Bit 4 .. 7:	0000 xxxx (reserved)
3	...	Data 1	
4	...	Data 2	
5	...	Data 3	
6	...	Data 4	
7	...	Data 5	
8	...	Data 6	

If less than 6 data bytes are transmitted, the valid data is always located at the beginning (for example, nnn = 4: Data 1 to Data 4 valid).

If the request was carried out successfully, the drive confirms the request in its response:

Response from Drive System (to Write Segment)

Byte	Value	Contents	
1	44_{hex} or 04_{hex}	Bit 0 .. 2:	xxxx x100 = extended services
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx00 xxxx = no request
		Bit 6:	x1xx xxxx – if handshake was previously 0
			x0xx xxxx – if handshake was previously 1
		Bit 7:	0xxx xxxx = no error
2	00_{hex}	Control byte = 00_{hex} (no meaning)	
3 .. 8	00_{hex}	Data = $0000\ 0000_{hex}$ (no meaning)	

9.3.4.5 Abort of Data Transmission by Drive System

The drive can abort the sequential data transmission by sending an error status message (bits 0 to 2 in service byte = 000_{bin}, bit 7 = 1).

Bytes 5 to 8 of the error status message contain the error code:

Abort of Sequential Data Transmission by the Drive System			
Byte	**Value**	**Contents**	
1	F0 $_{hex}$ or B0 $_{hex}$	Bit 0 .. 2:	xxxx x000 = no request
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx11 xxxx = 4 byte data in Data/Error field
		Bit 6:	x1xx xxxx = sends back handshake 1
			x0xx xxxx = sends back handshake 0
		Bit 7:	1xxx xxxx = request not carried out, error
2	...	Subindex	
3 .. 4	...	Index	
5 .. 8	...	Error code	

The slave aborts the data transmission because an error has occurred. Previously transmitted data is no longer valid:

- In a write request, the drive discards all previously transmitted data.
- In a read request, the master discards all previously transmitted data.

9.3.4.6 Abort of Data Transmission by Master

The master can also abort sequential data transmission at any time by setting bit 7 (status) in the service byte to 1 (extended services only):

Abort of Sequential Data Transmission by Master			
Byte	**Value**	**Contents**	
1	B4 $_{hex}$ or 84 $_{hex}$	Bit 0 .. 2:	xxxx x100 = extended services
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx00 xxxx (without meaning, as bit 7 = 1)
		Bit 6:	x1xx xxxx = sends back handshake 1
			x0xx xxxx = sends back handshake 0
		Bit 7:	1xxx xxxx = error
2	00 $_{hex}$	Control byte = 00 $_{hex}$ (no meaning)	
3 .. 8	00 $_{hex}$	Data = 0000 0000 $_{hex}$ (no meaning)	

The drive responds with an error status message. The error code is 0 because the error is not on the drive side:

Response from the Drive System (to Abort of Transmission by Master)			
Byte	**Value**	**Contents**	
1	F0 hex or B0 hex	Bit 0 .. 2:	xxxx x000 = no request
		Bit 3:	xxxx 0xxx (reserved)
		Bit 4 .. 5:	xx11 xxxx 4 byte data in Data/Error field
		Bit 6:	x1xx xxxx = sends back handshake 1
			x0xx xxxx = sends back handshake 0
		Bit 7:	1xxx xxxx = request not carried out
2	00 hex	Subindex = 00 hex (no meaning)	
3 .. 4	00 hex	Index = 0000 hex (no meaning)	
5 .. 8	00 hex	Error code = 0000 0000 hex (no error from drive system)	

When data transmission is aborted, all previously transmitted data is no longer valid:

- In a write request, the drive discards all previously transmitted data.
- In a read request, the master discards all previously transmitted data.

9.4 Error Codes

If an error occurs (status bit = 1), the ESR servo drive sends information describing the error in the Data/Error fields. The individual fields have the following meaning:

Parameter Channel		**Contents**
Byte	**Error**	
5	Error 1	Error class
6	Error 2	Error code
7	Error 3	Additional code (highest-value byte)
8	Error 4	Additional code (lowest-value byte)

The following error codes are defined:

Parameter Channel				**Meaning**
Error 1	**Error 2**	**Error 3**	**Error 4**	
Class	**Code**	**Additional Code**		
00 hex	00 hex	00 hex	00 hex	No error
06 hex	03 hex	00 hex	00 hex	No access authorization
06 hex	05 hex	00 hex	10 hex	Invalid request parameter
06 hex	05 hex	00 hex	11 hex	Invalid subindex
06 hex	05 hex	00 hex	12 hex	Data too long or too short
06 hex	07 hex	00 hex	00 hex	Object does not exist
06 hex	08 hex	00 hex	00 hex	Data types do not match
08 hex	00 hex	00 hex	20 hex	Request cannot be carried out

Parameter Channel				Meaning
Error 1	**Error 2**	**Error 3**	**Error 4**	
Class	**Code**	**Additional Code**		
08 $_{hex}$	00 $_{hex}$	00 $_{hex}$	21 $_{hex}$	Cannot be carried out due to local controller
08 $_{hex}$	00 $_{hex}$	00 $_{hex}$	22 $_{hex}$	Cannot be carried out due to device operating state
08 $_{hex}$	00 $_{hex}$	00 $_{hex}$	30 $_{hex}$	Out of value range
08 $_{hex}$	00 $_{hex}$	00 $_{hex}$	40 $_{hex}$	Collision with other values

10 Diagnose Alarm

Diagnose alarms are sent by the servo drive in case of incoming and outgoing events. Thus, they functionally correspond to the Profibus diagnose, however, their contents, is not compatible with the Profibus diagnose messages. For alarms to arrive in the controller parameter *activate diagnose alarm* must be activated for at least one module. Then, alarms are reported to the controller assigned to this module.

Diagnose alarms are sent as ExtChannelDiag of the PROTOCOL_SPECIFIC (value 80000000 $_{hex}$) type. As servo drive error codes are always 16-bit numbers, for Profinet, however, only values from 0000 $_{hex}$ to 7FFF $_{hex}$ are permitted in the ChannelDiag field of the diagnose message, the error code is divided on fields ChannelDiag (high-byte) and ExtChannelDiag (low-byte) of the diagnose message. Due to the fact that value 0000 $_{hex}$ is not permitted in field ExtChannelDiag, the low-byte in this field is additionally OR-linked with 0100 $_{hex}$. Example: Error code A011 $_{hex}$ is transmitted as A0 $_{hex}$ in the ChannelDiag field and 0111 $_{hex}$ (0011 $_{hex}$ or 0100 $_{hex}$) in the ExtChannelDiag field.

For all known error codes, entries (combinations of ChannelDiag and ExtChannelDiag) are stored in the GSDML file with a corresponding text so that the configuration tool can output both the fault code of the servo drive as well as a clear-text message of the error if supported by the tool (works e. g. in TIA V13).

11 Protocol Types

The Ethernet, IP, ARP, DCP, DHCP, RPC, RT, and IRT protocols are Profinet IO standard. Additionally, device manufacturers may implement other protocols.

The Servo Drives New Generation with Profinet IO interface support the named protocols and additionally the manufacturer-specific ESR protocol described in detail in the following.

11.1 ESR Protocol

As standard, the ESR protocol used by SPP Windows via TCP driver CommTCP runs on port 6668. Therefore, SPP Windows can communicate directly with the servo drive via CommTCP and Ethernet.

The ESR protocol essentially maps the interface of the SPP Windows DLL to a simple ASCII protocol. The client (PC) sends an ASCII command to the server (servo drive) which responds to it with an ASCII text.

Commands always start with a character, followed by no or several parameters in the form of an ASCII hexadecimal string. The command is terminated with a CR/LF character string. The response string always consists of an error code in the form of an ASCII hexadecimal string followed by no to several response data in the form of an ASCII hexadecimal string, too. The response string is terminated with a LF character.

The following commands are defined:

Command	Format	Result Data	Example / Remark

Read	**R**iiiissll
	
	iiii = index
	ss = subindex
	ll = length
		dd...
		object data (only if error code = 0)
			R60410002
			(read status word)
			Number of data corresponds to object length, byte order must be observed.

Write	**W**iiiisslldd...
	
	iiii = index
	ss = subindex
	ll = length
		none, just error code	W604000020F80
			(write value 0x800f $_{hex}$ into control word)
			Length of data must correspond with the stated object length, byte order must be observed.

Identify	**I**	3×16 data bytes, 16 bytes each for the *vendor*, *model*, and *revision* strings.	The first byte of each string contains the length of the string, bytes 2 .. 16 (max.) the string.
Determine endian mode	**M**	1 data byte: 00 = big endian (Motorola) 01 = little endian (Intel)	Determines the format in which the data bytes are transmitted with commands and results.

Command	Format	Result Data	Example / Remark
Determine error message	**E**eeeeeeee eeeeeeee = 32-bit error code for which the error message has to be determined	n data bytes of the error message string	E000000AC The error code does not contain 0 but the length of the result string. Byte order of parameter eeeeeeee always in Motorola format.
Determine interface	**F**	4 data bytes	Interface identification as 32-bit word, observe byte order.
Determine node ID	**N**	4 data bytes	Node ID as 32-bit word, observe byte order.

The response string has one of the following forms:

Variant	Format	Remark
16-bit error codes	eeeedd ... eeee = 16-bit error code dd ... = optional result data (see command table)	Except for command E, data are not supplied when the error code is not equal to 0000.
32-bit error codes	**X**eeeeeeeedd ... (prefix X) eeeeeeee = 32-bit error code dd ... = optional result data (see command table)	Except for command E, data are not supplied when the error code is not equal to 0000.

Notes for the protocol:

- The command parameters (index, error code) are always transmitted in big endian format (Motorola). In contrast to that, data bytes of read and write commands are transmitted either in big or in little endian format. Therefore, the transmission format should be determined using command M directly after the connection has been established. The data bytes have to be changed accordingly.

  Example: Value 0x800F $_{hex}$ is written on the control word using W60400002800F for mode 0 (big endian) and W604000020F80 for mode 1 (little endian).
- Commands have to be terminated with CR/LF because the internal evaluation starts with CR. Therefore, an end-of-line identification in UNIX format (LF, only) is not permitted. In contrast to that, the response string contains only an LF at the end of the line:
- For simple tests, telnet can be used (enter port 6668).

Special features of direct communication with the TCP server in the servo drive:

- The endian mode of the servo drive is always little endian (mode command always supplies 01).

- Independent of device configuration and possibly set bus names, commands F (determine interface) and N always supply the same consistent values.
- Command E supplies error messages in English language.

Index