# Distance Sensors

# R1000

SerialLink communication protocol

Protocol version 1.00

Your automation, our passion.

PEPPERL+FUCHS

# Contents

# 1 Introduction

The SerialLink communication protocol is a simple serial protocol which can be used to configure Pepperl+Fuchs distance sensors and to retrieve measurement data as well as device status information.

Protocol properties:

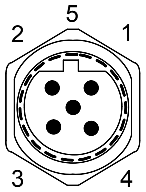
- Data is exchanged between one *sensor* and one *controller*.
- Data is exchanged in well defined frames.
- Data is transferred in (up to a certain extend) human readable ASCII characters.
- Frame checksums ensure data integrity (can be enabled optionally).

# 2 Protocol basics

## 2.1 Physical Layer

The SerialLink protocol is using a physical layer according to the RS-422 standard [1]. It specifies a full-duplex serial point-to-point connection with differential signaling. Serial data is transfered Byte-wise with 8 data bits, no parity and 1 stop bit (8N1) and a user-configured baud-rate. The most significant bit of a data byte is transfered first (MSB first).

R1000 devices use a B-coded M12 plug as RS-422 / SSI connector:




1 TXD+ (output)
2 TXD- (output)
3 RXD+ (input)
4 RXD- (input)
5 Shield

## 2.2 Communication Frames

The transfer of payload data between peers is organized in *frames*:

| 1 | 2 ... N-3 | N-2 N-1 | N |
|---|---|---|---|
| <STX> | Payload | Checksum | <ETX> |

Every frame starts with the marker STX (ASCII 0x02) and ends with the marker ETX (ASCII 0x03). In-between STX and ETX the actual payload (commands, parameter data, etc.) is inserted. A valid frame has a total size $N$ of at least 4 byte and must not exceed 500 byte for R1000 devices.

Payload data and the checksum are transported in a human-readable form using ASCII characters:

- two characters representing a single-byte hexadecimal value
- a sequence of digits representing a decimal value
- a sequence of characters representing a string

### Example

A simple command request for command ID '04' would look like this:

| 1 | 2 | 3 | 4 | |
|---|---|---|---|---|
| <STX> | '0' | '4' | <ETX> | ASCII characters |
| 0x02 | 0x30 | 0x34 | 0x03 | Hexadecimal values |

## 2.3 Frame Checksum

All communication frames optionally contain a two-byte checksum, which can be enabled or disabled by the user (see section 4.7). The checksum is calculated from the (binary) sum of all payload data bytes in the frame, i.e. excluding the leading STX and trailing ETX. The bitwise complement of the low-byte of this sum gives the checksum value:

$$\text{Checksum} = \left(\left(\sum Payload\right) \bmod 256\right) \text{ xor } 255$$

The checksum is inserted into the frame right before the ETX byte as ASCII encoded hexadecimal representation (i.e. as two additional data bytes as described above).

**Please note:**
Process data output in *Binary mode* contains the checksum value as a single byte instead of the standard two-byte ASCII representation. Refer to section 5.1 for a detailed description of the *Binary mode* process data format.

### Example

This example demonstrates the checksum calculation for an exemplary command frame (see section 2.4) which sets the 'Error delay' (parameter `'16'`) to a value of 79 ms:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|
| <STX> | '0' | '2' | '1' | '6' | '7' | '9' | <ETX> | ASCII characters |
| 0x02 | 0x30 | 0x32 | 0x31 | 0x36 | 0x37 | 0x39 | 0x03 | Hexadecimal values |
| | Command ID | | Parameter ID | | Parameter value | | | |

The payload data bytes `0x30, 0x32, 0x31, 0x36, 0x37, 0x39` of the frame sum up to `0x0139`. The modulo 256 of this sum gives `0x39`, whereof the binary inverted value `0xC6` is the checksum value. The hexadecimal values of the ASCII characters `'C6'` are `0x43 0x36` – which are the bytes that have to be inserted into the frame before `ETX`:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|
| <STX> | '0' | '2' | '1' | '6' | '7' | '9' | 'C' | '6' | <ETX> | ASCII characters |
| 0x02 | 0x30 | 0x32 | 0x31 | 0x36 | 0x37 | 0x39 | 0x43 | 0x36 | 0x03 | Hexadecimal values |
| | Payload | | | | | | Checksum | | | |

## 2.4 Command Frames

Requests sent from the user application to the sensor are called *command frames*. Each command frame consists of a mandatory *command ID* and optional command *arguments*:

| 1 | 2 | 3 | 4 | … | N-1 | N |
|---|---|---|---|---|---|---|
| <STX> | Command ID | | Command Arguments | | | <ETX> |

If checksum are enabled, command frames will also contain a checksum (see section 2.3 for details):

| 1 | 2 | 3 | 4 | … | N-3 | N-2 | N-1 | N |
|---|---|---|---|---|---|---|---|---|
| <STX> | Command ID | | Command Arguments | | | Checksum | | <ETX> |

Command frames have the following properties:

- Command frames always contain a (hexadecimal) *command ID*, which is always represented by two ASCII characters.
- Command frames may contain additional *command arguments*, which are also always represented by a sequence of ASCII characters.
- A valid command frame is always answered by a *data reply frame* (see section 2.5.1).
- An invalid command frame is always answered by an *error reply frame* (see section 2.5.2).

### Example

This example shows a command frame for an exemplary 'write parameter' command (CmdID `'02'`), which updates the parameter 'user tag location' (ID `'0C'`) to the value `'Door'`:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|
| <STX> | '0' | '2' | '0' | 'C' | 'D' | 'o' | 'o' | 'r' | <ETX> | ASCII characters |
| 0x02 | 0x30 | 0x32 | 0x30 | 0x43 | 0x44 | 0x6F | 0x6F | 0x72 | 0x03 | Hexadecimal values |
| | Command ID | | Parameter ID | | Parameter value | | | | | |

## 2.5 Reply Frames

The sensor answers an command frame (request) either with a *Data Reply Frame* or an *Error Reply Frame*.

### 2.5.1 Data Reply Frames

Data reply frames are sent if a command frame has been successfully processed. Each data reply frame consists of a *Reply ID* and optional additional *Reply Data*:

| 1 | 2 | 3 | 4 | ... | N-1 | N |
|---|---|---|---|---|---|---|
| <STX> | Reply ID | | Reply Data | | | <ETX> |

If checksum are enabled, data reply frames will also contain a checksum (see section 2.3 for details):

| 1 | 2 | 3 | 4 | ... | N-3 | N-2 | N-1 | N |
|---|---|---|---|---|---|---|---|---|
| <STX> | Reply ID | | Reply Data | | | Checksum | | <ETX> |

Data reply frames have the following properties:

- A data reply frame is only sent in response to a (valid) command frame.
- The reply ID of a data reply frames matches the command ID of the command frame, but with the MSB set (e.g. CmdID `'01'` will by answered with ReplyID `'81'`).
- Depending on the specific command, data reply frames may contain multiple additional data bytes. These data are usually represented as ASCII characters of the hexadecimal byte values.

#### Example

This example assumes that a 'get temperature' (CmdID `'05'`) command frame has been sent:

| 1 | 2 | 3 | 4 | |
|---|---|---|---|---|
| <STX> | '0' | '5' | <ETX> | ASCII characters |
| 0x02 | 0x30 | 0x35 | 0x03 | Hexadecimal values |
| | Command ID | | | |

As reply a frame with the ReplyID `'85'` and the temperature value `-12` is received:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | |
|---|---|---|---|---|---|---|---|
| <STX> | '8' | '5' | '-' | '1' | '2' | <ETX> | ASCII characters |
| 0x02 | 0x38 | 0x35 | 0x2D | 0x31 | 0x32 | 0x03 | Hexadecimal values |
| | Reply ID | | Reply Data | | | | |

### 2.5.2 Error Reply Frames

If an error occurred during the processing of a command frame (see section 2.4) an *error reply frame* is returned. It consists of a six letter *error code* surrounded by `STX` and `ETX`:

| 1 | 2 | ... | 7 | 8 |
|---|---|---|---|---|
| <STX> | Error Code | | | <ETX> |

If checksum are enabled, error reply frames will also contain a checksum (see section 2.3 for details):

| 1 | 2 | ... | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|
| <STX> | Error Code | | | Checksum | | <ETX> |

Error checking is performed at several stages which will produce specific error codes:

| # | error code | chksum | description |
|---|---|---|---|
| 1 | ERRFRM | 31 | Invalid frame received (e.g. frame too long) |
| 2 | ERRCHK | 40 | Valid frame received but checksum is invalid or missing |
| 3 | ERRSEQ | 2D | Valid frame received but processing of preceding request is not completed yet |
| 4 | ERRCMD | 42 | Valid frame received but command ID is invalid or unknown |
| 5 | ERRARG | 3C | Valid command received but required arguments are missing or invalid (e.g. unknown ParID) |
| 6 | ERRFBD | 4A | Valid command received but access is forbidden (e.g. write to read-only parameter) |
| 7 | ERRVAL | 33 | Valid command received but argument *value* is missing or invalid (e.g. value out of range) |
| 8 | ERRBSY | 28 | Valid command received but command could not be executed, try again later |
| | ERRNVM | 25 | Valid command received but command execution failed due to NVM access error |

### Example

The following command frame with an invalid CmdID '77':

| 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|
| <STX> | '7' | '7' | '9' | '1' | <ETX> | ASCII characters |
| 0x02 | 0x37 | 0x37 | 0x39 | 0x31 | 0x03 | Hexadecimal values |
| | Invalid CmdID | | Checksum | | | |

will be answered with an ERRCMD error reply frame:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|---|
| <STX> | 'E' | 'R' | 'R' | 'C' | 'M' | 'D' | '4' | '2' | <ETX> | ASCII characters |
| 0x02 | 0x45 | 0x52 | 0x52 | 0x43 | 0x4D | 0x44 | 0x34 | 0x32 | 0x03 | Hexadecimal values |
| | Error Code | | | | | | Checksum | | | |

## 2.6 Process Data Frames

The user can enable continuous output of process data, which is then provided by the sensor in dedicated *process data frames* with a configurable data format in ASCII or binary form. Please refer to chapter 5 for a detailed description of process data output and available configuration options.

### 2.6.1 ASCII Process Data Frames

Process data in ASCII formats (see section 5.1) is provided by the sensor in dedicated ASCII process data frames, which can be identified by the leading '#' tag character followed by 8 digits of ASCII process data:

| 1 | 2 | 3 … 10 | 11 |
|---|---|---|---|
| <STX> | '#' | ASCII Process Data | <ETX> |

If checksum are enabled, ASCII process data will also contain a two-digit checksum (see section 2.3 for details):

| 1 | 2 | 3 … 10 | 11 12 | 13 |
|---|---|---|---|---|
| <STX> | '#' | ASCII Process Data | Checksum | <ETX> |

### Example

This example shows a ASCII process data frame in *Decimal* format for an exemplary distance value of `12340` (1234 mm):

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| ASCII characters | <STX> | '#' | '0' | '0' | '0' | '1' | '2' | '3' | '4' | '0' | <ETX> |
| Hexadecimal values | 0x02 | 0x23 | 0x30 | 0x30 | 0x30 | 0x31 | 0x32 | 0x33 | 0x34 | 0x30 | 0x03 |
| | | Tag | Process Data | | | | | | | | |

## 2.6.2 Binary Process Data Frames

To minimize latency and maximize the throughput the sensor can optionally provide process data also in a binary format (see section 5.1) wrapped into compact binary process data frames:

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| <STX> | Binary Process Data | | | | <ETX> |

If checksum are enabled, binary process data will contain a *single-byte* checksum (see section 2.3 for details):

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| <STX> | Binary Process Data | | | | Chksum | <ETX> |

The total frame length of a binary process data frame adds up to 6 byte without checksum and 7 byte with checksum. Please refer to section 5.1.4 for a more detailed description of the combined binary process data output format.

> **Please note:**
> The payload of a binary process data may contain bytes of any value including the `STX` and `ETX` bytes. The receiver should not confuse payload data with these frame markers. It should ensure the frame integrity with a size check.

> **Please note:**
> Binary process data frames can be distinguished from other frames by the MSB bit of the first payload data byte: For binary process data frames this bit is always set (1).

> **Please note:**
> In contrast to all other SerialLink communication frames the binary process data frame carries the checksum as *single* byte.

### Example

This example shows a binary process data frame (with checksum) for an exemplary status value of `0x84` and an exemplary distance value of `123450` (`0x01E23A`):

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| 0x02 | 0x84 | 0x01 | 0xE2 | 0x3A | 0x5E | 0x03 |
| STX | Binary Process Data | | | | Chksum | ETX |

## 2.7 Value encoding

For the SerialLink communication protocol the following generic rules are applied:

- Command IDs are always encoded as a single-byte HEX value represented by two ASCII characters.
- Parameter IDs are always encoded as a single-byte HEX value represented by two ASCII characters.
- Parameter values are always encoded as multi-byte decimal value represented by ASCII characters.
  The value might have an additional leading character for the sign (either `'+'` or `'-'`).
- Strings shall only contain printable ASCII characters (range `0x20 ... 0x7F`).
- Strings are allowed to use UTF-8 encoding (but size limitations apply to the *byte* count).

## 2.8 Frame sequencing

The SerialLink communication protocol is designed as a *Single-Request-Response* [2] communication:

- The controller should send a new command frame only when it has received a reply frame from the sensor for the preceding command frame. Otherwise the sensor might respond with an `ERRSEQ` error reply frame (see section 2.5.2) without processing the command.
- The controller may send command frames to the sensor while the process data output is active. However, processing of the command frames might impair the output timing of process data (see also 5.3).

## 2.9 Timing information

Timing parameters relevant for the SerialLink communication protocol protocol:

- Power on delay: 10 s
- Command frame reception timeout: none

# 3 Commands

## 3.1 Command overview

A valid command frame from the controller always contains a command ID followed by additional arguments (if required). The following commands are available in R1000 devices:

| CmdID | Arguments | Description | Reply Data |
|---|---|---|---|
| '01' | ParID | Read parameter | Parameter value |
| '02' | ParID & Value | Write parameter | – |
| '04' | – | Get device status | Status flags (Hexadecimal) as defined in section 3.4 |
| '05' | – | Get device temperature | Current temperature (signed decimal) in [℃] |
| '07' | FormatID | Poll process data | – |
| '08' | – | Start process data output | – |
| '09' | – | Stop process data output | – |
| '0A' | – | Read all parameters | List of values of all available parameters |
| '0B' | ParID List | Write multiple parameters | – |
| '0F' | ResetKey | Load factory settings | – |

## 3.2 CmdID '01': Read parameter

The command '01' provides read access to a single parameter out of a variety of sensor parameters. Please refer to chapter 4 for a comprehensive list of available parameters. The parameter is specified by a numeric two-digit *ParID* (e.g. '08'), which is appended as an argument to the command '01' within the command frame.

The sensor responds to a valid read request with a data reply frame containing the ReplyID '81' as well as the current value of the requested parameter. A numeric value is provided as sequence of digits in decimal notation. A string value is provided as a sequence of printable ASCII characters (without a terminal NUL character).

### Example

The following request contains a read parameter command for parameter '12' (measurement offset):

| 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|
| <STX> | '0' | '1' | '1' | '2' | <ETX> | ASCII characters |
| 0x02 | 0x30 | 0x31 | 0x31 | 0x32 | 0x03 | Hexadecimal values |
| | CmdID | | ParID | | | |

This is answered with a data reply frame with the ReplyID '81' and the read parameter value -1234:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |
|---|---|---|---|---|---|---|---|---|---|
| <STX> | '8' | '1' | '-' | '1' | '2' | '3' | '4' | <ETX> | ASCII characters |
| 0x02 | 0x38 | 0x31 | 0x2D | 0x31 | 0x32 | 0x31 | 0x32 | 0x03 | Hexadecimal values |
| | Reply ID | | Parameter Value | | | | | | |

## 3.3 CmdID '02': Write parameter

The command '02' provides write access to a single parameter out of a variety of sensor parameters. Please refer to chapter 4 for a comprehensive list of available parameters. The parameter is specified by a numeric two-digit *ParID* (e.g. '0A'), which is appended as an argument to the command '02' within the command frame. Furthermore the new value to be written to the parameter is appended to the command frame behind the CmdID and ParID. Numeric values are expected as sequence of digits in decimal notation with an optional sign character. String values are expected as a sequence of printable ASCII characters (optionally terminated by a NUL character).

The sensor responds to a valid write request with a data reply frame containing the ReplyID '82'.

### Example

The following write parameter request changes parameter '12' (measurement offset) to value +987:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|
| <STX> | '0' | '2' | '1' | '2' | '+' | '9' | '8' | '7' | <ETX> | ASCII characters |
| 0x02 | 0x30 | 0x32 | 0x31 | 0x32 | 0x2D | 0x31 | 0x32 | 0x31 | 0x03 | Hexadecimal values |
| | CmdID | | ParID | | Parameter Value | | | | | |

A successful write operation is answered with a data reply frame with the ReplyID '82':

| 1 | 2 | 3 | 4 | |
|---|---|---|---|---|
| <STX> | '8' | '2' | <ETX> | ASCII characters |
| 0x02 | 0x38 | 0x32 | 0x03 | Hexadecimal values |
| | Reply ID | | | |

## 3.4 CmdID '04': Get device status

The command '04' provides the current status flags of the device as a single byte:

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| reserved (always 1) | Defect | Error | Warning | Substitute value | On target | Switching signal 2 | Switching signal 1 |

The status byte is provided in hexadecimal representation (e.g. '0xA8') by a data reply frame with ReplyID '84'.

### Example

The following request reads the current device status via command '04':

| 1 | 2 | 3 | 4 | |
|---|---|---|---|---|
| <STX> | '0' | '4' | <ETX> | ASCII characters |
| 0x02 | 0x30 | 0x34 | 0x03 | Hexadecimal values |
| | CmdID | | | |

The data reply frame with ReplyID '84' provides the current status flags as hexadecimal value '0x86':

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | |
|---|---|---|---|---|---|---|---|---|
| <STX> | '8' | '4' | '0' | 'x' | '8' | '6' | <ETX> | ASCII characters |
| 0x02 | 0x38 | 0x34 | 0x30 | 0x78 | 0x38 | 0x36 | 0x03 | Hexadecimal values |
| | Reply ID | | Device Status Byte | | | | | |

## 3.5 CmdID '05': Get device temperature

The command '05' reads the current device temperature. The command request is answered with a data reply frame containing the ReplyID '85' followed by the temperature (degree Celsius) in decimal notation (up to 3 digits). A negative temperature is indicated by an additional leading minus character '-'.

> **Please note:**
> The device temperature refers to the temperature within the device which is usually significantly higher than the current ambient temperature.

### Example

The following request reads the current device temperature via command '05':

| 1 | 2 | 3 | 4 | |
|---|---|---|---|---|
| <STX> | '0' | '5' | <ETX> | ASCII characters |
| 0x02 | 0x30 | 0x35 | 0x03 | Hexadecimal values |
| | CmdID | | | |

The data reply frame with ReplyID '85' provides the current temperature as decimal value '45':

| 1 | 2 | 3 | 4 | 5 | 6 | |
|---|---|---|---|---|---|---|
| <STX> | '8' | '5' | '4' | '5' | <ETX> | ASCII characters |
| 0x02 | 0x38 | 0x35 | 0x34 | 0x35 | 0x03 | Hexadecimal values |
| | Reply ID | | Temperature | | | |

## 3.6 CmdID '07': Request single process data

The command '07' returns the current process data. The output format can be specified as an optional argument – similar to the parameter 'PD output format' (ParID '54'):

| FormatID | Format | Process Data String | Example Output |
|---|---|---|---|
| 0 | Distance Decimal | 8 decimal digits: distance, no status | '#00098765' |
| 1 | Distance Hexadecimal | 8 hexadecimal digits: distance, no status | '#000181CD' |
| 2 | Combined Hexadecimal | 6 hexadecimal digits: distance, 2 hexadecimal digits: status | '#0181CD04' |

Please refer to section 5.1 for a detailed description of the different process data output formats. The binary format is not supported by command '07'.

The command request is answered with a data reply frame containing the ReplyID '87' and the process data in the requested output format and with the current measurement resolution (see section 4.3).

### Example

The following request polls a distance value in decimal format via command '07':

| 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|
| <STX> | '0' | '7' | '0' | <ETX> | ASCII characters |
| 0x02 | 0x30 | 0x37 | 0x30 | 0x03 | Hexadecimal values |
| | CmdID | | Format | | |

The data reply frame with ReplyID '87' returns the current distance value $01234567$:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| <STX> | '8' | '7' | '0' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | <ETX> | ASCII chars |
| 0x02 | 0x38 | 0x37 | 0x30 | 0x31 | 0x32 | 0x33 | 0x34 | 0x35 | 0x36 | 0x37 | 0x03 | Hex values |
| | Reply ID | | Process Data (distance value) | | | | | | | | | |

## 3.7 CmdID '08': Start process data output

The command '08' starts the continuous output of process data with the currently configured output format – see chapter 5 for details. The command request is answered with a data reply frame containing the ReplyID '88'. The process data itself is sent as a separate frame (see section 2.6 for details).

### Example

The following request starts the process data output via command '08':

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| ASCII characters | <STX> | '0' | '8' | <ETX> |
| Hexadecimal values | 0x02 | 0x30 | 0x38 | 0x03 |
| | | CmdID | CmdID | |

The data reply frame with ReplyID '88' signals a successful completion of the operation:

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| ASCII characters | <STX> | '8' | '8' | <ETX> |
| Hexadecimal values | 0x02 | 0x38 | 0x38 | 0x03 |
| | | Reply ID | Reply ID | |

## 3.8 CmdID '09': Stop process data output

The command '09' stops the continuous output of process data (if active). The successful, processing of the command request is acknowledged by a data reply frame containing the ReplyID '89'.

### Example

The following request stops the process data output via command '09':

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| ASCII characters | <STX> | '0' | '9' | <ETX> |
| Hexadecimal values | 0x02 | 0x30 | 0x39 | 0x03 |
| | | CmdID | CmdID | |

The request is answered with a data reply frame with ReplyID '89':

| | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| ASCII characters | <STX> | '8' | '9' | <ETX> |
| Hexadecimal values | 0x02 | 0x38 | 0x39 | 0x03 |
| | | Reply ID | Reply ID | |

Protocol version 1.00 Document release 2023-06

## 3.9 CmdID ’0A’: Read all parameters

The command ’0A’ reads the values of *all* available sensor parameters (see chapter 4). The command request is acknowledged by a data reply frame containing the ReplyID ’8A’ followed by a list of parameter values. Each list entry is composed of the ParID (two digits) followed by the actual parameter value (variable number of characters) and a terminal *newline* sequence (<CR><LF>).

A list of parameter values looks like this (with the parameter value highlighted in green for better readability):

```
01Pepperl+Fuchs<CR><LF>
02https://www.pepperl-fuchs.com<CR><LF>
03OMR150M-R1000-SSI-V1V1B<CR><LF>
[...]
302<CR><LF>
310<CR><LF>
325000<CR><LF>
3310000<CR><LF>
34100<CR><LF>
[...]
503<CR><LF>
513<CR><LF>
521<CR><LF>
530<CR><LF>
540<CR><LF>
551<CR><LF>
```

**Please note:**
If enabled, the frame checksum is calculated over *all* data bytes between STX and ETX (including the <CR> and <LF> characters) and is then inserted as a two-digits hexadecimal value right before the ETX symbol, just as in every frame (see section 2.3).

### Example

The following request reads all parameter values via command ’0A’:

| 1 | 2 | 3 | 4 | |
|---|---|---|---|---|
| <STX> | ’0’ | ’A’ | <ETX> | ASCII characters |
| 0x02 | 0x30 | 0x41 | 0x03 | Hexadecimal values |
| | CmdID | | | |

The request is answered with a data reply frame with ReplyID ’8A’:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|
| <STX> | ’8’ | ’A’ | ’0’ | ’1’ | ’P’ | ’+’ | ’F’ | <CR> | <LF> | ASCII characters |
| 0x02 | 0x38 | 0x41 | 0x30 | 0x31 | 0x50 | 0x2B | 0x46 | 0x0D | 0x0A | Hexadecimal values |
| | Reply ID | | Parameter ID | | Parameter value | | | Newline | | |

| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | ... | |
|---|---|---|---|---|---|---|---|---|---|---|
| ’0’ | ’3’ | ’R’ | ’1’ | ’0’ | ’0’ | ’0’ | <CR> | <LF> | ... | ASCII characters |
| 0x30 | 0x33 | 0x52 | 0x31 | 0x30 | 0x30 | 0x30 | 0x0D | 0x0A | ... | Hexadecimal values |
| Parameter ID | | Parameter value | | | | | Newline | | | |

| 81 | 82 | 83 | 84 | 85 | 86 | |
|---|---|---|---|---|---|---|
| ’5’ | ’5’ | ’1’ | <CR> | <LF> | <ETX> | ASCII characters |
| 0x35 | 0x35 | 0x31 | 0x0D | 0x0A | 0x03 | Hexadecimal values |
| Parameter ID | | Value | Newline | | | |

## 3.10 CmdID ’0B’: Write multiple parameters

The command ’0B’ enables the user to modify a *set* of sensor parameters at once. Please refer to chapter 4 for a list of modifiable sensor parameters. Similar to command ’0A’ the set of new parameter values is provided as a list of of parameter values, where each list entry is composed of the ParID (two digits) followed by the new parameter value (variable number of characters) and a terminal *newline* sequence (<CR><LF>). The command request is acknowledged by a data reply frame containing the ReplyID ’8B’.

> **Please note:**
> If the parameter list for command ’0B’ contains an error (e.g. a non-writable parameter) then the whole request is rejected by the device with an error reply frame (see section 2.5.2) and no parameter is changed.

> **Please note:**
> The output of command ’0A’ (except for the read-only identification parameters) can be used as input string for command ’0B’ in order to realize a simple backup and restore mechanism.

### Example

This example changes three measurement parameters en-bloc via command ’0B’:

- set ’measurement delay’ (ParID ’10’) to 6 ms (value ’2’)
- set ’measurement resolution’ (ParID ’11’) to 0.1 mm (value ’0’)
- set ’measurement offset’ (ParID ’12’) to –987 mm (value ’-9870’)

This translates into the following parameter list (with the parameter value highlighted in green for better readability):

```
102<CR><LF>
110<CR><LF>
12-9870<CR><LF>
```

The resulting command request looks like this:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | |
|---|---|---|---|---|---|---|---|---|---|---|
| <STX> | ’0’ | ’B’ | ’1’ | ’0’ | ’2’ | <CR> | <LF> | ’1’> | ’1’ | ASCII characters |
| 0x02 | 0x30 | 0x42 | 0x31 | 0x30 | 0x32 | 0x0D | 0x0A | 0x31 | 0x31 | Hexadecimal values |
| | Reply ID | | Parameter ID | | Value | Newline | | Parameter ID | | |

| 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | |
|---|---|---|---|---|---|---|---|---|---|---|
| ’0’ | <CR> | <LF> | ’1’ | ’2’ | ’-’ | ’9’ | ’8’ | ’7’ | ’0’ | ASCII characters |
| 0x30 | 0x0D | 0x0A | 0x31 | 0x32 | 0x2D | 0x39 | 0x38 | 0x37 | 0x30 | Hexadecimal values |
| Value | Newline | | Parameter ID | | Value | | | | | |

| 21 | 22 | 23 | |
|---|---|---|---|
| <CR> | <LF> | <ETX> | ASCII characters |
| 0x0D | 0x0A | 0x03 | Hexadecimal values |
| Newline | | | |

The request is answered with a data reply frame with ReplyID ’8B’:

| 1 | 2 | 3 | 4 | |
|---|---|---|---|---|
| <STX> | ’8’ | ’B’ | <ETX> | ASCII characters |
| 0x02 | 0x38 | 0x42 | 0x03 | Hexadecimal values |
| | Reply ID | | | |

## 3.11 CmdID '0F': Load factory settings

The command '0F' will revert all *writable* parameters to their individual default values. The command expects the string 'RESET' as argument, which works as additional validation key to confirm that a factory reset is indeed intended. The command request is answered with a data reply frame containing the ReplyID '8F'.

> **Please note:**
> A factory reset via SerialLink will *not* modify the basic settings for the serial communication, i.e. the parameters 'Serial interface mode' (ParID '50') and 'Serial baud-rate' (ParID '51') are *not* set to their default values.

### Example

The following request triggers a factory reset via command '0F':

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | |
|---|---|---|---|---|---|---|---|---|---|
| <STX> | '0' | 'F' | 'R' | 'E' | 'S' | 'E' | 'T' | <ETX> | ASCII characters |
| 0x02 | 0x30 | 0x46 | 0x52 | 0x45 | 0x53 | 0x45 | 0x54 | 0x03 | Hexadecimal values |
| | CmdID | | Validation Key | | | | | | |

The data reply frame with ReplyID '8F' signals a successful completion of the operation:

| 1 | 2 | 3 | 4 | |
|---|---|---|---|---|
| <STX> | '8' | 'F' | <ETX> | ASCII characters |
| 0x02 | 0x38 | 0x46 | 0x03 | Hexadecimal values |
| | Reply ID | | | |

Protocol version 1.00 Document release 2023-06

# 4 Parameters

Using the commands `'01'` and `'02'` the user may read and modify various parameter settings of the device (see chapter 3). This chapter describes all parameters available in R1000 devices. Default parameter values are printed **bold**.

## 4.1 Parameter types

The sensor provides access to different types of parameters. The following table gives an overview of the available types:

| type | description |
|---|---|
| `enum` | enumeration type with a set of numeric values |
| `int` | signed integer values |
| `uint` | unsigned integer values |
| `string` | strings composed of ASCII / UTF-8 characters (except control characters `0x00` ... `0x1F`) |

Regardless of their type, each parameter belongs to one of the following access groups:

| access | description |
|---|---|
| sRO | static Read-Only access (value never changes) |
| RO | volatile Read-Only access (value might change during operation) |
| RW | persistent Read-Write access (non-volatile storage) |
| vRW | volatile Read-Write access (lost on reset) |

Most sensor parameters are stored in non-volatile memory. Thus their value also persists a power-cycle of the device.

**Please note:**
Non-volatile storage has a limited number of write cycles only (300.000 cycles for R1000 devices). Therefore all non-volatile parameters should be written only if necessary.

## 4.2 Identification parameters

The parameter ID range `'01'...'0F'` contains various *identification* parameters:

| ID | type | description | values | access |
|---|---|---|---|---|
| `'01'` | `string` | Vendor name | up to 32 byte | sRO |
| `'02'` | `string` | Vendor text | up to 32 byte | sRO |
| `'03'` | `string` | Product name | up to 32 byte | sRO |
| `'04'` | `string` | Product ID (order number) | up to 32 byte | sRO |
| `'05'` | `string` | Product text | up to 32 byte | sRO |
| `'06'` | `string` | Serial number | up to 16 byte | sRO |
| `'07'` | `string` | Hardware revision | up to 8 byte | sRO |
| `'08'` | `string` | Firmware revision | up to 8 byte | sRO |
| `'09'` | `string` | Interface revision | up to 8 byte | sRO |
| `'0A'` | `string` | User tag 'application' | up to 32 byte | RW |
| `'0B'` | `string` | User tag 'function' | up to 32 byte | RW |
| `'0C'` | `string` | User tag 'location' | up to 32 byte | RW |

**Please note:**
The string parameters might contain any sequence of ASCII or UTF-8 characters up to the specified length. The control characters `0x00` ... `0x1F` are not allowed.

## 4.3 Measurement parameters

The parameter ID range '10'...'1F' provides various parameter for configuration of the distance *measurement* acquisition:

| ID | type | description | values | access |
|---|---|---|---|---|
| '10' | enum | Measurement delay | 0: 25 ms<br>1: 12 ms<br>2: 6 ms<br>**3: 3 ms** | RW |
| '11' | enum | Measurement resolution | 0: 0.1 mm<br>**1: 1 mm** | RW |
| '12' | int | Measurement offset | Offset value [0.1 mm]: –999.9999 m . . . 999.9999 m<br>Default: **0** (0 mm) | RW |
| '13' | enum | Counting direction | **0: Forward**<br>1: Reverse | RW |
| '14' | enum | Smart Hold | 0: Disabled<br>**1: Enabled** | RW |
| '15' | enum | Error substitution value | **0: Last valid value**<br>1: Replacement value 0<br>2: Replacement value -1 | RW |
| '16' | uint | Error delay | Delay value [1 ms]: 0 ms . . . 9999 ms<br>Default: **50** (50 ms) | RW |

## 4.4 Digital I/O parameters

The parameter ID range '20'...'2F' provides various parameters for configuration of the *digital I/O* signals I/Q1 and Q2:

| ID | type | description | values | access |
|---|---|---|---|---|
| '20' | enum | I/Q1 type | 1: Binary output (push-pull)<br>**4: Hi-Z**<br>5: Input (active-high)<br>6: Input (active-low) | RW |
| '21' | enum | I/Q1 output function | **2: Switching signal 1 (SSC1)**<br>4: Error<br>5: Error + Warning<br>255: Inactive (constant) | RW |
| '22' | enum | I/Q1 input function | **1: Emitter off** | RW |
| '23' | enum | I/Q1 polarity | **0: Active-high**<br>1: Active-low | RW |
| '25' | enum | Q2 type | 1: Binary output (push-pull)<br>**4: Hi-Z** | RW |
| '26' | enum | Q2 output function | **3: Switching signal 2 (SSC2)**<br>4: Error<br>5: Error + Warning<br>255: Inactive (constant) | RW |
| '28' | enum | Q2 polarity | **0: Active-high**<br>1: Active-low | RW |

## 4.5 Switching signals parameters

The parameter ID range `'30'...'3F'` provides various parameters for configuration of the *switching signal* channels SSC1 and SSC2:

| ID | type | description | values | access |
|---|---|---|---|---|
| '30' | enum | SSC1 mode | 0: Deactivated (constant)<br>1: Single point<br>**2: Window** | RW |
| '31' | enum | SSC1 logic | **0: Normal**<br>1: Inverted | RW |
| '32' | uint | SSC1 setpoint 1 | Distance value [0.1 mm]: 0 m . . . 999.9999 m<br>Default: **5000** (500 mm) | RW |
| '33' | uint | SSC1 setpoint 2 | Distance value [0.1 mm]: 0 m . . . 999.9999 m<br>Default: **10000** (1 m) | RW |
| '34' | uint | SSC1 hysteresis | Hysteresis value [0.1 mm]: 0 m . . . 999.9999 m<br>Default: **100** (10 mm) | RW |
| '38' | enum | SSC2 mode | 0: Deactivated (constant)<br>1: Single point<br>**2: Window** | RW |
| '39' | enum | SSC2 logic | **0: Normal**<br>1: Inverted | RW |
| '3A' | uint | SSC2 setpoint 1 | Distance value [0.1 mm]: 0 m . . . 999.9999 m<br>Default: **10000** (1 m) | RW |
| '3B' | uint | SSC2 setpoint 2 | Distance value [0.1 mm]: 0 m . . . 999.9999 m<br>Default: **200000** (20 m) | RW |
| '3C' | uint | SSC2 hysteresis | Hysteresis value [0.1 mm]: 0 m . . . 999.9999 m<br>Default: **100** (10 mm) | RW |

## 4.6 User interface parameters

The parameter ID range `'40'...'4F'` provides various parameters for configuration of the *user interface* (HMI):

| ID | type | description | values | access |
|---|---|---|---|---|
| '40' | enum | Display language | **0: English**<br>1: German | RW |
| '41' | enum | Display orientation | **0: Normal (0°)**<br>1: Rotated (180°) | RW |
| '42' | enum | Display timeout (screensaver) | **1: 5 min**<br>2: 15 min<br>3: 30 min | RW |

## 4.7 Serial interface parameters

The parameter ID range `'50'...'5F'` provides various parameters for configuration of the *serial interface*:

| ID | type | description | values | access |
|---|---|---|---|---|
| `'50'` | `enum` | Serial interface mode | 0: disabled<br>**1: SSI Binary**<br>2: SSI Gray<br>3: SerialLink (RS-422) | RW |
| `'51'` | `enum` | Serial baud-rate | 0: 4800 baud<br>1: 9600 baud<br>2: 19 200 baud<br>**3: 38 400 baud**<br>4: 115 200 baud | RW |
| `'52'` | `enum` | SSI error bit | 0: Substitution value<br>**1: Substitution value + error**<br>2: Substitution value + error + warning | RW |
| `'53'` | `enum` | SerialLink frame checksum (see section 2.3) | **0: Disabled**<br>1: Enabled | RW |
| `'54'` | `enum` | SerialLink PD format (see section 5.1) | **0: Decimal (Distance only)**<br>1: Hexadecimal (Distance only)<br>2: Combined (Distance + Status)<br>3: Binary (Status + Distance) | RW |
| `'55'` | `enum` | SerialLink PD autostart (see section 5.2) | **0: Disabled**<br>1: Enabled | RW |

**Please note:**
Changing the serial interface mode or the serial baud-rate will affect the SerialLink connection directly!

# 5 Process data

Process data can be either polled via command `'07'` or it can be continuously transmitted. The continuous output is started using command `'08'` (see section 3.7) and stopped using command `'09'` (see section 3.8). The process data output itself can be customized regarding *output format*. Additionally the continuous process data output can be started automatically after power-on using the *autostart* option. This chapter provides a detailed description of available options.

## 5.1 Process data formats

Process data can be transmitted in several formats as requested by the argument of command `'07'` or as configured by the parameter `'54'` (see also section 4.7). Process data always contains the current distance value and optionally the current device status (see also section 3.4). The following table lists all available process data formats along with an exemplary output (distance = 98765, status = 0x84):

| Format | Process Data String | Example Output |
|---|---|---|
| Distance Decimal | tag and 8 decimal digits: distance, no status | `'#00098765'` |
| Distance Hexadecimal | tag and 8 hexadecimal digits: distance, no status | `'#000181CD'` |
| Combined Hexadecimal | tag and 6 hexadecimal digits: distance, 2 hexadecimal digits: status | `'#0181CD84'` |
| Combined Binary | 1 (binary) byte: status, 3 (binary) bytes: distance | `0x84 0x01 0x81 0xCD` |

**Please note:**
The *resolution* of the distance value can be configured using parameter `'11'` (see 4.3)

### 5.1.1 Distance Decimal PD format

This PD format provides ASCII process data frames containing the current distance reading in decimal representation with 8 digits:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| <STX> | '#' | '1' | '2' | '3' | '4' | '5' | '6' | '7' | '8' | <ETX> | ASCII characters |
| 0x02 | 0x23 | 0x31 | 0x32 | 0x33 | 0x34 | 0x35 | 0x36 | 0x37 | 0x38 | 0x03 | Hexadecimal values |
| | Tag | Distance Value (decimal) | | | | | | | | | |

If checksums are enabled, the frame will contain an additional two-digit checksum string before the `ETX` (see section 2.6.1).

### 5.1.2 Distance Hexadecimal PD format

This PD format provides ASCII process data frames containing the current distance reading in hexadecimal representation with 8 digits:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| <STX> | '#' | 'A' | 'B' | '1' | '2' | 'C' | 'D' | '3' | '4' | <ETX> | ASCII characters |
| 0x02 | 0x23 | 0x41 | 0x42 | 0x31 | 0x32 | 0x43 | 0x44 | 0x33 | 0x34 | 0x03 | Hexadecimal values |
| | Tag | Distance Value (hexadecimal) | | | | | | | | | |

If checksums are enabled, the frame will contain an additional two-digit checksum string before the `ETX` (see section 2.6.1).

### 5.1.3 Combined Hexadecimal PD format

This PD format provides ASCII process data frames containing the current distance reading in hexadecimal representation with 6 digits followed by the current device status flags in hexadecimal representation with 2 digits:

| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| <STX> | '#' | 'A' | 'B' | 'C' | 'D' | 'E' | 'F' | '8' | 'A' | <ETX> | ASCII characters |
| 0x02 | 0x23 | 0x41 | 0x42 | 0x43 | 0x44 | 0x45 | 0x46 | 0x38 | 0x41 | 0x03 | Hexadecimal values |
| | Tag | Distance Value (hexadecimal) | | | | | | Status (hexadecimal) | | | |

If checksums are enabled, the frame will contain an additional two-digit checksum string before the ETX (see section 2.6.1).

### 5.1.4 Combined Binary PD format

This PD format provides binary process data frames containing the current device status flags in binary representation with 1 byte followed by the current distance reading in binary representation with 3 bytes:

| 1 | 2 | 3 | 4 | 5 | 7 |
|---|---|---|---|---|---|
| 0x02 | 0x8A | 0xAB | 0xCD | 0xEF | 0x03 |
| STX | Status | Distance Value | | | ETX |

If checksums are enabled, the frame will contain an additional single-byte checksum value before the ETX (see section 2.6.2).

## 5.2 Process data autostart

Continuous process data output normally needs to be explicitly started with a command '08'. With the optional *autostart* functionality process data output is started automatically after power-on of the device, without the need of a command '08' request. The automatically started process data output can be stopped using command '09' at any time.

The autostart is enabled or disabled using parameter '55' (see section 4.7).

**Please note:**
When autostart is enabled, the continuous process data output after power-on will lead to continuous traffic on the serial interface. Therefore the user interface will be locked, i.e. no parameters can be changed via the HMI. In particular, the autostart option cannot be disabled again via HMI, unless PD output is stopped via SerialLink using command '09' or a factory reset is performed via HMI or SerialLink.

## 5.3 Process data timing

The time interval between two process data frames in *continuous* mode depends on the availability of new measurement data as well as the maximum transmission speed (baud-rate) of the serial communication. The following table lists the process data output interval (time between the STX bytes of consecutive process data frames) for different baud-rates:

| PD Output | 4800 baud | 9600 baud | 19 200 baud | 38 400 baud | 115 200 baud |
|---|---|---|---|---|---|
| **ASCII modes** | 34 ms | 18 ms | 10 ms | 6 ms | 3 ms |
| **Binary mode** | 17 ms | 9 ms | 5 ms | 3 ms | 1 ms |

**Please note:**
These timing values should not be confused with the *measurement delay* of the sensor, which is determined by parameter '10' (see section 4.3).

# A Migrating from VDM100 RS-422 to R1000 SerialLink

The R1000 SerialLink protocol specified in this document is based on the VDM100 RS-422 protocol specified in [3]. Please note, that the R1000 SerialLink protocol is *not backwards-compatible* to the VDM100 RS-422 protocol. This appendix gives a short overview of major differences of the protocol implementations. Client applications designed for the VDM100 RS-422 protocol need to be updated for communication the R1000 SerialLink protocol.

## A.1 Command overview

The following table provides a quick reference for mapping VDM100 RS-422 commands to R1000 SerialLink commands:

| VDM100 Command | R1000 Command | Remarks |
|---|---|---|
| '01': Laser pointer on | n/a | R1000 has no laser pointer |
| '02': Laser pointer off | n/a | R1000 has no laser pointer |
| '03': Laser pointer auto | n/a | R1000 has no laser pointer |
| '04': Read status | '04': Get device status | R1000 and VDM100 status flags differ |
| '05': Read temperature | '05': Get device temperature | R1000 outputs decimal temperature value |
| '06': Read release number | '01' with ParID '08'/'09' | R1000 revision strings have different format |
| '07': Load factory settings | '0F': Load factory settings | R1000 command requires additional argument |
| '08': Start measurement | '08': Start process data output | R1000 command start continuous output |
| '09': Stop measurement | '09': Stop process data output | |
| '0A': Read configuration | '0A': Read all parameters | |
| '10': Resolution | '01'/'02' with ParID '11' | See section 4.3 |
| '11': Offset | '01'/'02' with ParID '12' | See section 4.3 |
| '12': Counting direction | '01'/'02' with ParID '13' | See section 4.3 |
| '13': Measured value age | '01'/'02' with ParID '10' | See section 4.3 |
| '14': Set output format | '01'/'02' with ParID '54' | See section 4.7 |
| '15': Set output mode | n/a | Mode 'individually' replaced by new command '07' |
| '16': Freeze at v=0 | '01'/'02' with ParID '14' | See section 4.3 |
| '17': Error substitute value | '01'/'02' with ParID '15' | See section 4.3 |
| '18': Error delay | '01'/'02' with ParID '16' | See section 4.3 |
| '20': I/O1 Input or output | '01'/'02' with ParID '20' | See section 4.4 |
| '21': I/O1 Output function | '01'/'02' with ParID '21' | See section 4.4, available functions differ |
| '22': I/O1 Output polarity | '01'/'02' with ParID '23' | See section 4.4 |
| '23': I/O1 Input function | '01'/'02' with ParID '22' | See section 4.4, available functions differ |
| '22': I/O1 Input polarity | '01'/'02' with ParID '23' | See section 4.4 |
| '28': I/O2 Input or output | '01'/'02' with ParID '25' | See section 4.4 |
| '29': I/O2 Output function | '01'/'02' with ParID '26' | See section 4.4, available functions differ |
| '2A': I/O2 Output polarity | '01'/'02' with ParID '28' | See section 4.4 |
| '2B': I/O2 Input function | n/a | R1000 has no input on Q2 |
| '2C': I/O2 Input polarity | n/a | R1000 has no input on Q2 |
| '30': Threshold position 1 | '01'/'02' with ParID '32'/'3A' | See section 4.5 |
| '31': Threshold position 2 | '01'/'02' with ParID '33'/'3B' | See section 4.5 |
| '32': Threshold speed | n/a | R1000 has no output function 'speed' |
| '40': Activate checksum | '01'/'02' with ParID '53' | See section 4.7 |
| '41': Set serial protocol | n/a | R1000 implements SerialLink protocol only |
| '50': Set multiple parameters | '0B': Write multiple parameters | |

## A.2 Notable protocol differences

The R1000 SerialLink protocol has been enhanced in several aspects compared to the VDM100 RS-422 protocol:

- The option to protect communication frames with a checksum is no longer applied to *reply frames* only but does now also affect *command frames*. Please refer to section 2.3 for further details.
- Error handling for command frames has been significantly enhanced. A malformed request is now answered with an *error reply frame* providing a much more specific error code. Please refer to section 2.5.2 for more details.
- Modifications to device parameters are no longer handled by parameter-specific commands. Instead parameter access has been unified by introducing a 'read parameter' command (cmdID `'01'`) and a 'write parameter' command (cmdID `'02'`). Please refer to chapter 4 for details and a complete list of available parameters.
- Polling a single process data value from the sensor is now done with a dedicated 'poll process data' command (cmdID `'07'`) instead of configuring a *'single'* process data output mode. Please refer to chapter 5 for details.
- Continuous process data output is now using dedicated *process data frames* instead of a data reply frame. Please refer to section 2.6 for further details.
- The protocol now offers an *autostart* option to start continuous process data output automatically after power-up. Please refer to section 5.2 for further details.
- The SerialLink protocol is using decimal or hexadecimal representation of values more consistently:
  - Command IDs are specified in hexadecimal representation
  - Parameter IDs are specified in hexadecimal representation
  - Parameter values are specified in decimal representation
  - Command `'04'` returns the device status in hexadecimal representation
  - Command `'05'` returns the device temperature in decimal representation

# References

[1] *ANSI/TIA/EIA-422-B Electrical Characteristics of Balanced Voltage Differential Interface Circuits*, also known as *ITU-T Recommendation T-REC-V.11* or *X.27*
http://www.itu.int/rec/T-REC-V.11/en

[2] *Transport Message Exchange Pattern: Single-Request-Response*, W3C, 2001-09
http://www.w3.org/2000/xp/Group/1/10/11/2001-10-11-SRR-Transport_MEP

[3] *Manual Distance Measurement Device VDM100/G2*, Pepperl+Fuchs, P+F, 2013-08

# Your automation, our passion.

## Explosion Protection

- Intrinsic Safety Barriers
- Signal Conditioners
- FieldConnex® Fieldbus
- Remote I/O Systems
- Electrical Ex Equipment
- Purge and Pressurization
- Industrial HMI
- Mobile Computing and Communications
- HART Interface Solutions
- Surge Protection
- Wireless Solutions
- Level Measurement

## Industrial Sensors

- Proximity Sensors
- Photoelectric Sensors
- Industrial Vision
- Ultrasonic Sensors
- Rotary Encoders
- Positioning Systems
- Inclination and Acceleration Sensors
- Fieldbus Modules
- AS-Interface
- Identification Systems
- Displays and Signal Processing
- Connectivity

**Pepperl+Fuchs Quality**
Download our latest policy here:

**www.pepperl-fuchs.com/quality**

#70060 clean – 2023-06-01 13:54

PEPPERL+FUCHS

**www.pepperl-fuchs.com**



DOCT-8730