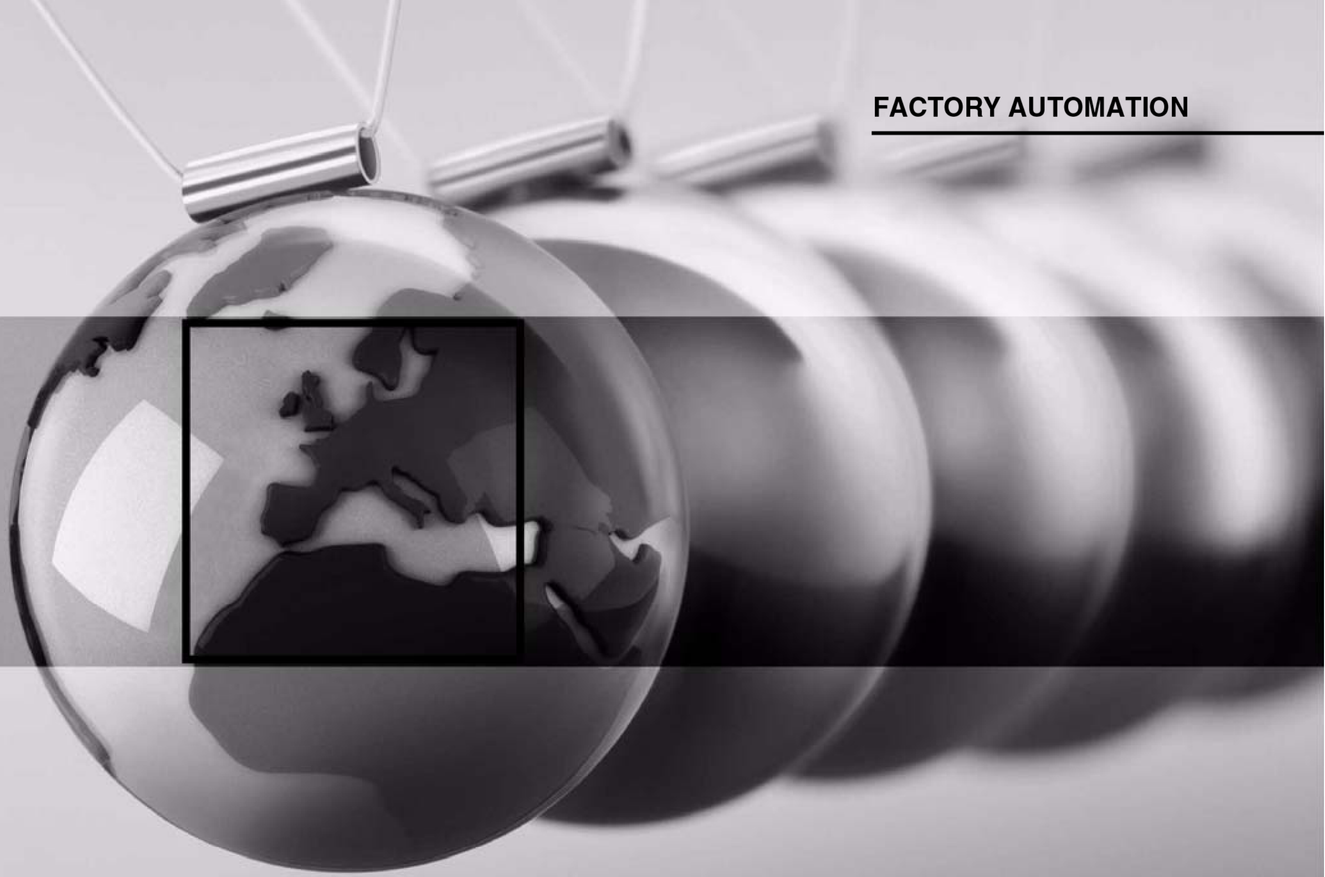
FACTORY AUTOMATION

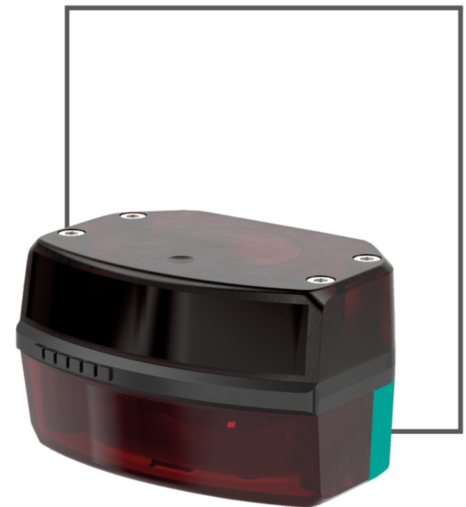


OPERATION INSTRUCTIONS

OMDxxx–R2300

Ethernet communication protocol

Protocol version 1.05



PEPPERL+FUCHS

SENSING YOUR NEEDS

Contents

1 Protocol basics 5
1.1 Basic design . . . 5
1.2 HTTP command protocol . . . 5
1.2.1 Sending commands . . . 5
1.2.2 Query argument encoding . . . 6
1.2.3 Replies to commands . . . 6
1.2.4 HTTP/1.1 persistent connections . . . 7
1.2.5 HTTP request and reply – low level example . . . 7
1.2.6 HTTP status codes . . . 7
1.2.7 Sensor error codes . . . 8
1.2.8 Protocol information (get_protocol_info) . . . 8

2 Sensor parametrization using HTTP 10
2.1 Parameter types . . . 10
2.1.1 Enumeration values (enum) . . . 10
2.1.2 Boolean values (bool) . . . 10
2.1.3 Bit fields (bitfield) . . . 11
2.1.4 Integer values (int, uint) . . . 11
2.1.5 Double values (double) . . . 11
2.1.6 String values (string) . . . 11
2.1.7 IPv4 address and network mask values (IPv4) . . . 12
2.1.8 NTP timestamp values (ntp64) . . . 12
2.1.9 Collection of values (array) . . . 12
2.2 Commands for sensor parametrization . . . 13
2.2.1 list_parameters – list parameters . . . 13
2.2.2 get_parameter – read a parameter . . . 14
2.2.3 set_parameter – change a parameter . . . 14
2.2.4 reset_parameter – reset a parameter to its default value . . . 14
2.2.5 reboot_device – restart the sensor firmware . . . 15
2.2.6 factory_reset – reset the sensor to factory settings . . . 15
2.3 Basic sensor information . . . 17
2.3.1 Parameter overview . . . 17
2.3.2 Device family (device_family) . . . 17
2.3.3 User defined tag (user_tag) . . . 17
2.4 Sensor capabilities . . . 18
2.4.1 Parameter overview . . . 18
2.4.2 Device features (feature_flags) . . . 18
2.4.3 Emitter type (emitter_type) . . . 18
2.4.4 Layer configuration (layer_count and layer_inclination) . . . 18
2.5 Ethernet configuration . . . 19
2.5.1 Parameter overview . . . 19
2.5.2 IP address mode (ip_mode) . . . 19
2.6 Measuring configuration . . . 20
2.6.1 Parameter overview . . . 20
2.6.2 Mode of operation (operating_mode) . . . 20
2.6.3 Scan rate (scan_frequency, scan_frequency_measured) . . . 20
2.6.4 Scan direction (scan_direction) . . . 21
2.6.5 Scan resolution (samples_per_scan) . . . 21
2.6.6 Scan acquisition sector (measure_start_angle / measure_stop_angle) . . . 22
2.6.7 Layer configuration (layer_enable) . . . 22
2.7 Alignment aid (pilot laser) . . . 22
2.7.1 Parameter overview . . . 22
2.7.2 Pilot laser activation (pilot_laser) . . . 22
2.7.3 Pilot laser sector (pilot_start_angle and pilot_stop_angle) . . . 23
2.8 User interface configuration . . . 23
2.8.1 Parameter overview . . . 23
2.8.2 Locator indication (locator_indication) . . . 23

PEPPERL+FUCHS

2.9 System status . . . 24
2.9.1 Parameter overview . . . 24
2.9.2 System status flags (status_flags) . . . 25

3 Scan data output using UDP **26**
3.1 Principles of scan data acquisition . . . 26
3.1.1 Sensor coordinate system . . . 26
3.1.2 Scan data coordinate system . . . 27
3.1.3 Distance readings . . . 27
3.1.4 Echo amplitude readings . . . 27
3.1.5 Timestamps . . . 28
3.2 Principles of scan data output . . . 29
3.2.1 Introduction . . . 29
3.2.2 Scan data connection handles . . . 29
3.2.3 Scan data output customization . . . 29
3.3 Commands for managing scan data output . . . 31
3.3.1 `request_handle_udp` – request for an UDP-based scan data channel . . . 31
3.3.2 `release_handle` – release a data channel handle . . . 32
3.3.3 `start_scanoutput` – initiate output of scan data . . . 32
3.3.4 `stop_scanoutput` – terminate output of scan data . . . 33
3.3.5 `set_scanoutput_config` – reconfigure scan data output . . . 33
3.3.6 `get_scanoutput_config` – read scan data output configuration . . . 34
3.4 Transmission of scan data . . . 36
3.4.1 Basic packet structure . . . 36
3.4.2 Typical structure of a scan data header . . . 37
3.4.3 Scan data header status flags . . . 38
3.4.4 Scan data packet type C1 – distance and amplitude . . . 39
3.5 Data transmission using UDP . . . 40

4 Filter-based scan data processing **41**
4.1 Introduction to scan data filtering . . . 41
4.1.1 Block-wise processing . . . 41
4.1.2 Moving-window processing . . . 41
4.1.3 Filter processing at scan edges . . . 42
4.2 Filter algorithms . . . 42
4.2.1 No filter (pass-through) . . . 42
4.2.2 Average filter . . . 43
4.2.3 Median filter . . . 43
4.2.4 Maximum filter . . . 43
4.3 Filter configuration . . . 43
4.3.1 Parameter overview . . . 44
4.3.2 Filter types (filter_type) . . . 44
4.3.3 Filter width (filter_width) . . . 44
4.3.4 Filter error handling (filter_error_handling) . . . 45
4.3.5 Maximum filter margin (filter_maximum_margin) . . . 45

5 Advanced topics **46**
5.1 Device discovery using SSDP . . . 46
5.1.1 SSDP search request . . . 46
5.1.2 SSDP device reply . . . 46
5.1.3 SSDP device description . . . 47

A Migrating from R2000 to R2300 **48**
A.1 Functional comparison . . . 48
A.2 PFSDP command implementation . . . 48
A.2.1 Commands available on R2000 devices only . . . 48
A.2.2 Commands available on R2300 devices only . . . 48
A.2.3 Error handling for command requests . . . 48
A.3 PFSDP parameter implementation . . . 49
A.3.1 Measurement configuration . . . 49
A.3.2 Pilot laser . . . 49
A.3.3 User tag and user notes . . . 49
A.3.4 HMI parameters . . . 49
A.4 PFSDP scan data implementation . . . 50
A.4.1 Connection handling . . . 50

A.4.2 Scan data output customization . . . 50
A.4.3 Scan data packet types . . . 50
A.5 Filter-based scan data processing . . . 50

B Troubleshooting the Ethernet communication **51**
B.1 Checking the Ethernet setup . . . 51
B.2 Debugging using a web browser . . . 51
B.3 Debugging using Wireshark . . . 51

C Protocol version history **53**
C.1 Protocol version 1.05 . . . 53

D Document change history **54**
D.1 Release 2022-08 (protocol version 1.05) . . . 54
D.2 Release 2020-10 (protocol version 1.05) . . . 54

Index for commands and parameters **55**

References **56**

1 Protocol basics

This chapter describes the basics of the Pepperl+Fuchs scan data protocol (PFSDP).

1.1 Basic design

The communication protocol specification is based on the following basic design decisions:

- A simple command protocol using HTTP requests (and responses) is provided in order to parametrize and control the sensor. The HTTP can be accessed using a standard web browser or by establishing temporary TCP/IP connections to the HTTP port.
- Sensor process data (scan data) is received from the sensor using a separate UDP/IP channel. The UDP channel provides an efficient transmission of scan data combined with minimal latency.

Output of scan data is always conform to the following conventions:

- Data output is performed as packets with a packet size adapted to the common Ethernet frame size .
- A single packet always contains data of a single continuous scan only. Scan data output always starts with a new packet for every (new) scan.
- For scan data output the user application can select from multiple data types with different levels of information detail. This way a client can decide to receive only the amount of data needed for its individual application – reducing traffic. Furthermore this provides an easy way to implement future extensions to the scan data output (e.g. adding additional information) as well.
- The byte order for all binary data is *Little Endian* (least significant byte first). The DSP of the sensor and PC CPUs both use Little Endian – thus no conversions need to take place.
- The sensor capabilities restrict the number of active (concurrent) client connections. This does not imply though that the device can handle multiple concurrent connections with the maximum amount of (scan) data. Basically it is the users responsibility to design his (client) system or application in a way that the sensor can handle the amount of requested data without getting overloaded.

1.2 HTTP command protocol

The HTTP command protocol provides a simple unified way to control sensor operation for application software. HTTP commands are used to configure sensor measurement as well as to read and change sensor parameters. Furthermore it is used to set up (parallel) UDP data channels providing sensor scan data.

This section describes the basic HTTP command protocol and various commands available to the user. Transmission of scan data using an additional UDP channel is explained in section 3.4.

1.2.1 Sending commands

Sending commands to the sensor is done using the Hypertext Transfer Protocol (HTTP) as defined by RFC 2616 [5]. Each HTTP command is constructed as **Uniform Resource Identifier** (URI) according to RFC 3986 [8] with the following basic structure:

```
<scheme>:<authority>/<path>?<query>#<fragment>
```

A typical HTTP request to the sensor looks like:

```
http://<sensor IP address>/cmd/<cmd_name>?<argument1=value>&<argument2=value>
```

Thus, in terms of an URI a valid HTTP command is composed of the following parts:

- **scheme** is always 'http://'
- **authority** is represented by the IP address of the sensor (and a port number, if necessary)
- **path** consists of the prefix 'cmd/' and the name of the requested command ('<cmd_name>')
- **query** lists additional arguments for the specific command
- **fragment** is currently not used – anything following the hash mark will be ignored

Please note:
The order of the command arguments (within <query>) is interchangeable at will. Sole exception is the argument handle (see section 3.3), which has to be specified always first in order to identify the client scan data connection – provided that this is required for the requested command.

Please note:
The maximum length of a HTTP request URI is limited to 255 B. Any request with a longer URI will be rejected with a HTTP status code 400 (see section 1.2.6). Typical user application usually do not exceed this limitation, though.

1.2.2 Query argument encoding

The query part of the command URI (see section 1.2.1) is used to transport additional arguments to HTTP commands (compliant to RFC 3986 [8]). This section describes the composition of arguments as "key=value" pairs.

HTTP command arguments are composed using the following scheme ("key=value" pairs):

```
key=value[;value][&key=value]
```

The key denotes an argument that receives one or more values. Multiple values for a single argument are separated by a semicolon ';'. A single command takes multiple arguments separated by an ampersand '&'.

Please note:
Some characters are reserved within an URI and need to be percent encoded according to the rules of RFC 3986 [8]. Most notably, if parameter values contain URI delimiters like the question mark '?', equal sign '=' or the ampersand '&', these characters need to be escaped on the URI.

1.2.3 Replies to commands

After sending a command to the sensor the following replies can be received:

- **HTTP status code**
  A HTTP command will be answered with a standard HTTP status code first. This code indicates whether the command (i.e. URI) is known and has been received correctly. An error code is returned only if the URI is invalid or could not be processed. Please refer to section 1.2.6 for a detailed description of HTTP status codes used by the R2300.
- **Command error code**
  Normally the HTTP status code is read as 'OK'. In this case the result of the command processing can be evaluated using two return values: error_code and error_text. error_code contains a numeric result code for the command call, while error_text provides a textual error description. Both values are returned using JSON encoding [10]. Section 1.2.7 provides a detailed description of all R2300 command error codes.
- **Command reply data**
  The body of a command reply contains any requested payload data. This data is always transmitted using JSON encoding [10]. Large amounts of data might be output using base64 encoded JSON arrays.

Please note:
The character encoding used for all JSON encoded command replies of the R2300 is always UTF-8 (RFC 7159 [10]).

Please note:
If the R2300 is not able to pre-compute the Content-Length of the response, it might choose to use "Transfer-encoding: chunked" where the length information comes as hex byte count numbers interleaved with the actual content. This can only happen when using HTTP/1.1 persistent connections (see section 1.2.4) and HTTP client libraries should be able to interpret the possibly chunked response data as needed. Alternatively the application can choose to use HTTP/1.0 requests only in order to avoid chunked responses.

1.2.4 HTTP/1.1 persistent connections

The R2300 provides full support for HTTP/1.1 as specified by RFC 2616 [5] including persistent connections. In order to establish a persistent connection a HTTP request should include the `"Connection: keep-alive"` header. The R2300 will respond and usually keep the connection open except in some rare cases.

Please note:
HTTP client libraries should be able to automatically add the required header for using a single persistent connection. Proper configuration of the library for this use case is out of the scope of this documentation.

1.2.5 HTTP request and reply – low level example

This section shows an example, how a HTTP request is transmitted to the sensor without using a web-browser. Lets assume, that the following HTTP request shall be send:

```
http://<sensor IP address>/cmd/get_parameter?list=scan_frequency
```

This request is translated into a simple string (using HTTP/1.0 in this example):

```
GET /cmd/get_parameter?list=scan_frequency HTTP/1.0\r\n\r\n
```

This string is then send as payload data of a TCP/IP packet to the sensor. The sensor then sends back a TCP/IP packet with the HTTP reply as payload data. The HTTP reply can be parsed as simple text string with the following content:

```
HTTP/1.0 200 OK\r\n
Expires: -1\r\n
Pragma: no-cache\r\n
Content-Type: text/plain\r\n
Connection: close\r\n
\r\n
{\r\n
"scan_frequency":50,\r\n
"error_code":0,\r\n
"error_text":"success"\r\n
}\r\n
```

The most important parts of this HTTP reply are the first line containing the HTTP error code and the last few lines containing the requested information wrapped within a single JSON-encoded [10] object denoted by a pair of curly brackets.

Please note:
It is highly recommended to use a third party HTTP library instead of a new custom implementation. Standards-compliant HTTP client implementations are widely available for most operation systems and hardware platforms (e.g. `Libwww` [12] or `libcURL` [13]).

1.2.6 HTTP status codes

The following table lists all HTTP status codes used by the device:

status code	message	description
200	`OK`	request successfully received
400	`Bad Request`	wrong URI syntax or URI too long
403	`Forbidden`	permission denied for this URI
404	`Not Found`	unknown command code or unknown URI
405	`Method not allowed`	invalid method requested (currently only `GET` is allowed)

Examples for (invalid) queries causing a HTTP error

request	status code	error message
`http://<ip>/cmd/nonsense`	400	"unrecognized command"
`http://<ip>/cmd/get_parameter&test`	400	"unrecognized command"
`http://<ip>/cmd/get_parameter?list`	400	"parameter without value"
`http://<ip>/test`	404	"file not found"
`http://<ip>/test/`	404	"file not found"
`http://<ip>/test/file`	404	"file not found"

1.2.7 Sensor error codes

The following table lists some generic error codes (`error_code`) returned by the device:

error code	description (`error_text`)
0	"success"
100	"unknown argument '%s'"
110	"unknown parameter '%s'"
120	"invalid handle or no handle provided"
130	"required argument '%s' missing"
200	"invalid value '%s' for argument '%s'"
210	"value '%s' for parameter '%s' out of range"
220	"write-access to read-only parameter '%s'"
230	"insufficient memory"
240	"resource already/still in use"
333	"(internal) error while processing command '%s'"

Please note:
The error message in `error_text` might slightly vary depending on the firmware version and the specific error condition of the actual command. Do not expect to receive error messages exactly as listed above.

Examples for (invalid) commands provoking sensor error codes

command (query)	code	error message
/cmd/get_protocol_info?list=test	100	"Unknown argument 'list'"
/cmd/get_parameter?list=test	110	"Unknown parameter 'test'"
/cmd/start_scanoutput	120	"Invalid handle or no handle provided"
/cmd/start_scanoutput?handle=test	120	"Invalid handle or no handle provided"
/cmd/set_parameter?ip_address=777	200	"Invalid value '777' for argument 'ip_address'."
/cmd/set_parameter?scan_frequency=999	210	"Value '999' for parameter 'scan_frequency' is out of range."
/cmd/set_parameter?serial=123456	220	"Write-access to read-only parameter 'serial'"

1.2.8 Protocol information (`get_protocol_info`)

Ethernet protocol users should be aware that depending on the protocol version some commands might not be available or might show different behavior. For this reason user applications should always check the protocol version using the dedicated command `get_protocol_info` which returns basic version information on the communication protocol:

parameter name	type	description
protocol_name	string	Protocol name (currently always 'pfsdp')
version_major	uint	Protocol major version (e.g. 1 for 'v1.02', 3 for 'v3.10')
version_minor	uint	Protocol minor version (e.g. 2 for 'v1.02', 10 for 'v3.10')
commands	string	List of all available HTTP commands

This document refers to protocol version '1.05' which is implemented by R2300 firmware version '1.00' and newer.

Please note:
The command `get_protocol_info` will return the above information on every sensor – regardless of its firmware version. All other commands and their return values might be changed by updates to the communication protocol, though. Therefore it is strongly recommended to check the protocol version first.

Example

Query: `http://<sensor IP address>/cmd/get_protocol_info`

Reply:


```
{
"protocol_name":"pfsdp",
"version_major":1,
"version_minor":4,
"commands":[
"get_protocol_info",
"list_parameters",
"get_parameter",
"set_parameter",
"reset_parameter",
"factory_reset",
"reboot_device",
"request_handle_udp",
"release_handle",
"start_scanoutput",
"stop_scanoutput",
"get_scanoutput_config",
"set_scanoutput_config",
"feed_watchdog"
],
"error_code":0,
"error_text":"success"
}
```

2 Sensor parametrization using HTTP

2.1 Parameter types

The sensor provides access to different types of parameters. The following table gives a quick overview of the relevant types, a more detailed description follows in separate sub-sections:

type	description
`enum`	enumeration type with a set of named values (strings)
`bool`	boolean values (`on` / `off`)
`bitfield`	a set of boolean flags
`int`	signed integer values
`uint`	unsigned integer values
`double`	floating point values with double precision
`string`	strings composed of UTF-8 characters
`ipv4`	Internet Protocol version 4 addresses or network masks
`ntp64`	NTP timestamp values
`array`	collection of values of the same type

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

access	description
sRO	static Read-Only access (value never changes)
RO	Read-Only access (value might change during operation)
RW	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 (typically > 10.000 cycles). Therefore all non-volatile parameters should be written only if necessary.

2.1.1 Enumeration values (`enum`)

Notes on parameters using enumeration values (`enum`):

- An enumeration type parameter accepts a single value out of a list of predefined values.
- Each enumeration values is defined by a string ('named' value).
- Each enumeration value is typically (but not necessarily) unique to the specific parameter.
- Each `enum` parameter can hold only a single value at a time.
- URI: Named enumeration values use non-reserved ASCII characters only and need no percent encoding [8] when specified as argument to a command on the URI.

2.1.2 Boolean values (`bool`)

Notes on parameters using boolean values (`bool`):

- Boolean parameters are a special case of enumeration parameters.
- Only the named values `on` and `off` are accepted.
- Each `bool` parameter can hold only a single value at a time.

2.1.3 Bit fields (`bitfield`)

Notes on parameters using bit fields (`bitfield`):

- Bit fields combine multiple boolean flags into an unsigned integer value.
- Each flag occupies a single bit of the integer.
- Not every bit of the integer needs to be assigned to a flag.
- Bits might be marked as *reserved*. These should always be zero.
- Bit field parameters are read and written using the integer representation.

2.1.4 Integer values (`int`, `uint`)

Notes on parameters using signed integer values (`int`) and unsigned integer value (`uint`):

- Unless denoted differently, the value range of integer values is limited to 32 bit.
- Leading zeros are accepted when writing a value (they will be ignored).
- Neither a hexadecimal nor an octal representation of integer values is supported.

2.1.5 Double values (`double`)

Notes on parameters using double precision floating point values (`double`):

- A dot '`.`' is used as decimal mark (separating the decimal part from the fractional part of a `double` number).
- The floating point decimal format (xxx.yyy) should be used when accessing `double` parameters. The floating point exponential format (xxx.yyy Ezzz) is not supported.
- The number of significant digits of the fractional part of a `double` value might be limited for some parameters. Excess digits are rounded or discarded.

2.1.6 String values (`string`)

Notes on parameters using string values (`string`):

- Strings represent a set characters.
- All characters of the string need to be encoded in UTF-8 format [7].
- The maximum size of a string is usually limited. Please refer to the description of the specific parameter for its actual size limitation.
- URI: For write access to a `string` parameter, its new value is implicitly delimited by the surrounding '`=`' and '`&`' within the URI (see RFC 3986 [8]). Any additionally added delimiter (e.g. '`"`') will be interpreted as part of the string.
- URI: Some characters are reserved within an URI and need to be percent encoded [8] (see section 1.2.2 for details).

When parsing a `string`-typed parameter within an UTF-8 encoded command URI the sensor performs the following steps:

1. Dissect the URI into it individual parts
2. Resolve percent encoded characters
3. Check string for a valid UTF-8 encoding
4. Process the string (UTF-8 bytes), e.g. store it into non-volatile memory

When the sensor outputs a `string`-typed parameter in JSON format, it applies escaping for the following reserved UTF-8 characters (as required by RFC 7159 [10] section 2.5):

Protocol version 1.05 Document release 2022-08

PEPPERL+FUCHS

character	code	replacement
backspace	U+0008	'\b'
tabulator	U+0009	'\t'
new line	U+000A	'\n'
form feed	U+000C	'\f'
carriage return	U+000D	'\r'
double quote	U+0022	'\"'
solidus	U+002F	currently not replaced
backslash	U+005C	'\\'
other control characters	U+0000 ... U+001F	'\uXXXX'

2.1.7 IPv4 address and network mask values (IPv4)

Notes on parameters using IPv4 network addresses and subnet masks (IPv4):

- Addresses and network masks need to follow the rules of the Internet Protocol specification (RFC-791 [1])
- Addresses are denoted as string values in human-readable *dotted decimal* notation (i.e. 10.0.10.9)
- Subnet masks are denoted as string values in human-readable *dotted decimal* notation (i.e. 255.255.0.0)

2.1.8 NTP timestamp values (ntp64)

Notes on parameters using NTP timestamps (ntp64):

- NTP timestamps are part of the Network Time Protocol (NTP) as defined by RFC 1305 [2].
- NTP timestamps are represented as a 64 bit unsigned fixed-point integer number (uint64) in seconds in reference to a specific point in time. The most significant 32 bit represent the integer part (seconds), the lower 32 bit the fractional part.
- *Absolute* timestamps (synchronized time) refer to the time elapsed since 1 January 1900.
- *Relative* timestamps (raw system time) refer to the time elapsed since power-on of the sensor.

Please refer to section 3.1.5 for more details on timestamps.

2.1.9 Collection of values (array)

Notes on parameters using a collection of values (array):

- Parameters of type array store multiple *elements* of the same type. The type of the elements is specific to the individual parameter. See the description of the specific parameter for details.
- For array parameters both the number of elements as well as the value of each element is checked. The correct number of elements and allowed element values depend on the specific parameter.
- Read access to a array parameter returns its value as a encoded JSON array with the value for each element.
- Write access to a array parameter requires the array elements to be provided as a comma-separated list of values.
- Individual access to a specific element of an array parameter is not supported.

2.2 Commands for sensor parametrization

This section describes all commands available for manipulation of global sensor parameters.

2.2.1 `list_parameters` – list parameters

The command `list_parameters` returns a list of all available global sensor parameters.

Example

Query: `http://<sensor IP address>/cmd/list_parameters`

Reply:


```
{
"parameters":[
"ip_mode",
"ip_address",
"subnet_mask",
"gateway",
"mac_address",
"ip_mode_current",
"ip_address_current",
"subnet_mask_current",
"gateway_current",
"serial",
"vendor",
"product",
"part",
"revision_fw",
"revision_hw",
"feature_flags",
"emitter_type",
"device_family",
"max_connections",
"scan_frequency_min",
"scan_frequency_max",
"radial_range_min",
"radial_range_max",
"radial_resolution",
"angular_resolution",
"angular_fov",
"sampling_rate_min",
"sampling_rate_max",
"up_time",
"power_cycles",
"operation_time",
"operation_time_scaled",
"temperature_current",
"temperature_min",
"temperature_max",
"system_time_raw",
"status_flags",
"scan_frequency_measured",
"locator_indication",
"pilot_laser",
"pilot_start_angle",
"pilot_stop_angle",
"user_tag",
"layer_enable",
"scan_frequency",
"scan_direction",
"measure_start_angle",
"measure_stop_angle",
"operating_mode",
"samples_per_scan"
],
"error_code":0,
"error_text":"success"
}
```

2.2.2 `get_parameter` – read a parameter

The command `get_parameter` reads the current value of one or more global sensor parameters:

```
http://<sensor IP address>/cmd/get_parameter?list=<param1>;<param2>
```

Command arguments

- `list` – semicolon separated list of parameter names (optional)

If the argument `list` is not specified the command will return the current value of all available parameters.

Example

Query: `http://<sensor IP address>/cmd/get_parameter?list=scan_frequency;scan_frequency_measured`

Reply:
```
{
"scan_frequency":50,
"scan_frequency_measured":49.900000,
"error_code":0,
"error_text":"success"
}
```

2.2.3 `set_parameter` – change a parameter

Using the command `set_parameter` the value of any write-accessible global sensor parameter can be changed:

```
http://<sensor IP address>/cmd/set_parameter?<param1>=<value>&<param2>=<value>
```

Command arguments

- `<param1> = <value>` – new <value> for parameter <param1>
- `<param2> = <value>` – new <value> for parameter <param2>
- ...

Please note:
The command `set_parameter` returns an error message, if any parameter specified as command argument is unknown or a read-only parameter. The return values `error_code` and `error_text` have appropriate values in this case (see section 1.2.7).

Example

Query: `http://<sensor IP address>/cmd/set_parameter?scan_frequency=50`

Reply:
```
{
"error_code":0,
"error_text":"success"
}
```

2.2.4 `reset_parameter` – reset a parameter to its default value

The command `reset_parameter` resets one or more global sensor parameters to their factory default values:

```
http://<sensor IP address>/cmd/reset_parameter?list=<param1>;<param2>
```

Command arguments

- `list` – semicolon separated list of parameter names (optional)

> **Please note:**
> If the argument `list` is not specified the command will load the factory default value for all parameters writeable with `set_parameter`!

> **Please note:**
> This command applies to global R/W parameters accessible via the command `set_parameter` only. If the argument `list` contains an unknown or a read only parameter, an error message will be returned.

> **Please note:**
> Resetting a parameter to its default value might require a device restart in order to take effect. For example, this applies to all Ethernet configuration parameters (see section 2.5).

Example

Query: `http://<sensor IP address>/cmd/reset_parameter?list=scan_frequency;scan_direction`

Reply:
```
{
"error_code":0,
"error_text":"success"
}
```

2.2.5 `reboot_device` – restart the sensor firmware

The command `reboot_device` triggers a soft reboot of the sensor firmware:

```
http://<sensor IP address>/cmd/reboot_device
```

Command arguments

The command accepts no additional arguments. The reboot is performed shortly after the HTTP reply has been sent.

> **Please note:**
> A reboot terminates all running scan data output. All scan data handles are invalidated and have to be renewed from scratch after reboot (see section 3.4).

> **Please note:**
> A device reboot takes up to 60 s (depending on the sensor configuration). The reboot is completed as soon as the sensor answers to HTTP command requests again and the system status flag *Initialization* (see section 2.9.2) is cleared.

Example

Query: `http://<sensor IP address>/cmd/reboot_device`

Reply:
```
{
"error_code":0,
"error_text":"success"
}
```

2.2.6 `factory_reset` – reset the sensor to factory settings

The command `factory_reset` performs a complete reset of all sensor settings to factory defaults and reboots the device. Its result is similar to a call of `reset_parameter` without any arguments followed by a call to `reboot_device`.

Command arguments

The command accepts no additional arguments. The factory reset and device reboot is performed shortly after the HTTP reply has been sent.

Please note:
The factory reset performs a device reboot, because some changes take effect at sensor boot time only (e.g. all changes to Ethernet configuration parameters – see section 2.5).

Example

Query: `http://<sensor IP address>/cmd/factory_reset`

Reply:
```
{
"error_code":0,
"error_text":"success"
}
```

Protocol version 1.05 Document release 2022-08

PEPPERL+FUCHS

2.3 Basic sensor information

This section describes all sensor parameters which are available to the user.

2.3.1 Parameter overview

The following table lists numerous parameters (mostly read-only) which provide basic sensor information:

parameter name	type	description	access
device_family	uint	Numeric unique identifier (see below)	sRO
vendor	string	Vendor name (max. 100 chars)	sRO
product	string	Product name (max. 100 chars)	sRO
part	string	Part number (max. 32 chars)	sRO
serial	string	Serial number (max. 32 chars)	sRO
revision_fw	string	Firmware revision (max. 32 chars)	sRO
revision_hw	string	Hardware revision (max. 32 chars)	sRO
user_tag	string	User defined name (max. 32 bytes)	RW

These entries are comparable to generic information available on IO-Link devices. In contrast to IO-Link most strings have no size limitation, though. Furthermore each parameter can be read individually using the command get_parameter.

2.3.2 Device family (device_family)

The parameter device_family can be used to identify compatible device families. A single device family is defined as group of devices with identical functionality (regarding the Ethernet protocol). This identifier can be used to check if the connected device is compatible with the client application (e.g. DTM user interface).

Currently the following values are defined for device_family:

value	name	description
0	reserved	never used
1	OMDxxx-R2000	R2000 OMD UHD raw data devices with ultra-high resolution
2	OBDxxx-R2000	R2000 OBD detection devices with standard features
3	OMDxxx-R2000-HD	R2000 OMD HD raw data devices with high resolution
4	reserved	
5	OMDxxx-R2300	R2300 OMD multi-line scanner
6	OMDxxx-R2000-SD	R2000 OMD SD raw data devices with standard resolution
7	OMDxxx-R2300	R2300 OMD single-line scanner

2.3.3 User defined tag (user_tag)

The parameter user_tag is a string that can be used by the user to store information about the device. It can store any data of type string (see section 2.1) with a length of up 32 bytes.

The default value for user_tag is typically a short version of the product name (parameter product).

Please note:
The string stored in user_tag must not contain NUL bytes.

PEPPERL+FUCHS

2.4 Sensor capabilities

2.4.1 Parameter overview

The following static read-only parameters describe the sensor capabilities:

parameter name	type	unit	description	access
feature_flags	array		sensor feature flags (see below)	sRO
emitter_type	uint		type of light emitter used by the sensor (see below)	sRO
radial_range_min	double	m	min. measuring range (distance)	sRO
radial_range_max	double	m	max. measuring range (distance)	sRO
radial_resolution	double	m	*mathematical* resolution of distance values in scan data output	sRO
angular_fov	double	°	max. angular field of view	sRO
angular_resolution	double	°	*mathematical* resolution of angle values in scan data output	sRO
layer_count	uint		number of distinct scan planes (layers)	sRO
layer_inclination	array	°	array with inclinations of individual scan planes (layers)	sRO
scan_frequency_min	double	Hz	min. supported scan rate (see section 2.6)	sRO
scan_frequency_max	double	Hz	max. supported scan rate (see section 2.6)	sRO
sampling_rate_min	uint	Hz	min. supported sampling rate (see section 2.6)	sRO
sampling_rate_max	uint	Hz	max. supported sampling rate (see section 2.6)	sRO
max_connections	uint		max. number of concurrent scan data channels (connections)	sRO

2.4.2 Device features (feature_flags)

The parameter feature_flags returns a JSON [10] encoded list of features available for the queried device. Currently the following features are defined:

feature name	description	reference	PFSDP version
ethernet	Ethernet interface		v1.00 or newer
scan_data_filter	Block-wise filtering of scan data	Chapter 4	v1.03 or newer
scan_data_filter_moving	Moving-window filtering of scan data	Chapter 4	v1.05 or newer

If a feature is available, its name is listed within the feature_flags array.

2.4.3 Emitter type (emitter_type)

The parameter emitter_type can be used to determine the type of light emitter (aka transmitter) used by the specific sensor. Currently the following emitter types are defined for R2300 devices:

type	description
0	undefined / reserved
1	not available
2	infrared laser (905 nm)

2.4.4 Layer configuration (layer_count and layer_inclination)

The parameters layer_count and layer_inclination provide information on the layer configuration of R2300 devices. The following table lists typical layer configurations:

Device family	layer_count	layer_inclination
R2300 OMD multi-line scanner	4	$[-4.5°; -1.5°; +4.5°; +1.5°]$
R2300 OMD single-line scanner	4	$[0°; 0°; 0°; 0°]$

Please note:
The array layer_inclination specifies the inclination of subsequent scans in the exact output order.

2.5 Ethernet configuration

2.5.1 Parameter overview

The following parameters allow configuration changes of the Ethernet interface:

parameter	type	description	access	default
ip_mode	enum	IP address mode: static, dhcp, autoip	RW	autoip
ip_address	ipv4	static IP mode: sensor IP address	RW	10.0.10.76
subnet_mask	ipv4	static IP mode: subnet mask	RW	255.0.0.0
gateway	ipv4	static IP mode: gateway address	RW	0.0.0.0
ip_mode_current	enum	current IP address mode: static, dhcp, autoip	RO	autoip
ip_address_current	ipv4	current sensor IP address	RO	169.254.x.y
subnet_mask_current	ipv4	current subnet mask	RO	255.255.0.0
gateway_current	ipv4	current gateway address	RO	0.0.0.0
mac_address	string	sensor MAC address ("000D81xxxxxx")	sRO	–

The read-only parameters ip_mode_current, ip_address_current, subnet_mask_current and gateway_current provide access to the currently active IP configuration. This is especially useful when using automatic IP configuration via DHCP or AutoIP.

Please note:
Any changes to the Ethernet configuration (using set_parameter or reset_parameter) are applied after a system reboot only! The command reboot_device (see section 2.2.5) is available to initiate a reboot using the Ethernet protocol.

2.5.2 IP address mode (ip_mode)

The parameter ip_mode configures one of the following IP address modes:

IP mode	description
static	static IP configuration using ip_address, subnet_mask, gateway
autoip	automatic IP configuration using "Zero Configuration Networking" [16]
dhcp	automatic IP configuration using a DHCP server

Please note:
With automatic IP configuration using DHCP or AutoIP the parameters ip_address_current and subnet_mask_current might return the invalid IP address 0.0.0.0, if no valid IP address has been assigned to the sensor yet (e.g. if no DHCP server is found).

Please note:
If dhcp mode has been configured but no DHCP server is available R2300 devices will fall back to autoip mode while continuing to search for a DHCP server in the background.

2.6 Measuring configuration

2.6.1 Parameter overview

The following (global) parameters are available for basic measurement configuration:

parameter name	type	unit	description	access	default
operating_mode	enum	–	mode of operation: measure, emitter_off	vRW	measure
scan_frequency	double	1 Hz	scan rate (50 Hz / 100 Hz)	RW	100 Hz
scan_direction	enum	–	direction of rotation: ccw only	RW	ccw
samples_per_scan	uint	samples	number of readings per scan	RO	501
scan_frequency_measured	double	1 Hz	measured scan rate (current value)	RO	–
layer_enable	array	–	enable individual layers: on, off	RW	on,on,on,on
measure_start_angle	int	1/10 000°	start angle for measurements (scan acquisition)	RW	-500000
measure_stop_angle	int	1/10 000°	stop angle for measurements (scan acquisition)	RW	+500000

2.6.2 Mode of operation (operating_mode)

The parameter operating_mode controls the mode of operation of the sensor. Currently, the following modes are available:

operating mode	description
measure	Sensor is recording scan data
emitter_off	Emitter is disabled, no scan data is recorded

The mode measure is the normal mode of operation of the sensor and default after power-on. The mode emitter_off allows the user to deactivate the light emitter, e.g. to avoid interference with other optical devices. A mode switch from measure to emitter_off can only be performed, if no scan data connections are active, i.e. all handles have been released. While the operating mode is set to emitter_off, no new scan data connection handles can be requested (see section 3.2). This state is also signaled by the system status flag scan_output_muted (see section 2.9.2).

Please note:
The parameter operating_mode is a non-persistent parameter, i.e. it reverts to its default value after reboot, power cycle or factory reset.

Example

Query: http://<sensor IP address>/cmd/set_parameter?operating_mode=measure

Reply:
```
{
"error_code":0,
"error_text":"success"
}
```

2.6.3 Scan rate (scan_frequency, scan_frequency_measured)

The parameter scan_frequency defines the number of scans recorded per second (see section 3.1 for details). This is also called *scan rate*. For R2300 devices this value indirectly translates into the rotational speed of the internal mirror. Applications can select a scan rate of either 50 Hz or 100 Hz (default).

The parameter scan_frequency_measured reads back the actual scan rate resulting from the current rotational speed of the internal mirror with a resolution of 0.1 Hz. It is a read-only parameter.

Please note:
For R2300 devices the configured scan rate implicitly also defines the scan resolution (see section 2.6.5).

PEPPERL+FUCHS

Example

Query: `http://<sensor IP address>/cmd/get_parameter?list=scan_frequency;scan_frequency_measured`

Reply:
```
{
"scan_frequency":100,
"scan_frequency_measured":99.900000,
"error_code":0,
"error_text":"success"
}
```

2.6.4 Scan direction (`scan_direction`)

The parameter `scan_direction` defines the direction of rotation of the measuring beam while acquiring measurement data. R2300 devices are recording scan data always in counter-clockwise (`ccw`).

> **Please note:**
> This parameter is only available for compatibility with R2000 devices. Any attempt to set `scan_direction` to `cw` will result in an error message.

Example

Query: `http://<sensor IP address>/cmd/set_parameter?scan_direction=ccw`

Reply:
```
{
"error_code":0,
"error_text":"success"
}
```

2.6.5 Scan resolution (`samples_per_scan`)

The parameter `samples_per_scan` defines the number of samples recorded within a scan (for details please refer to section 3.1). This value implicitly also defines the *scan resolution*, i.e. the angular step between two subsequent measurements.

For R2300 devices the scan resolution directly depends on the configured scan rate (parameter `scan_frequency`, see section 2.6.3) and the internal *sampling rate* (this is also called *pulse repetition rate* in laser-safety terminology):

- At 100 Hz the sensor will acquire up to 501 samples per scan with a scan resolution of 0.2°.
- At 50 Hz the sensor will acquire up to 1001 samples per scan with a scan resolution of 0.1°.

The actual value additionally depends on the current measuring field of view, which is configured using the parameters `measure_start_angle` and `measure_stop_angle` (see section 2.6.6). The value of `samples_per_scan` always reflects the actual number of samples derived from the current configuration.

> **Please note:**
> On R2300 devices the parameter `samples_per_scan` is *read-only*. Any write access to this parameter will result into an error message.

Example

Query: `http://<sensor IP address>/cmd/get_parameter?list=samples_per_scan`

Reply:
```
{
"samples_per_scan":501,
"error_code":0,
"error_text":"success"
}
```

2.6.6 Scan acquisition sector (measure_start_angle / measure_stop_angle)

The angular range defined by the parameters measure_start_angle and measure_stop_angle determines the area, where the sensor is actively acquiring scan data. Outside this area the emitter is not activated. Using these parameters applications can limit the scanned area to a specific region of interest, if necessary. Per default the sensor acquires scan data for its maximum field of view (see parameter angular_fov in section 2.4.1).

Please note:
Restricting the measuring field of view using measure_start_angle and measure_stop_angle will result in a reduced number of measurements in a scan as reported by samples_per_scan (see section 2.6.5).

Please note:
The number of scan points that are output via a scan data connection can be further reduced using the connection-specific parameters start_angle and max_num_points_scan (see section 3.3.5 for details) as well as by applying a decimating scan data filter (see chapter 4 for details).

Example

Query: http://<sensor IP address>/cmd/set_parameter?measure_start_angle=0&measure_stop_angle=450000

Reply:
```
{
    "error_code":0,
    "error_text":"success"
}
```

2.6.7 Layer configuration (layer_enable)

The parameter layer_enable allows the application to enable (on) or disable (off) measurements for individual layers. If a layer is disabled, scan data for that layer is neither recorded nor output via an active scan data connection. The settings for the individual layers are specified as elements of an array.

Example

Query: http://<sensor IP address>/cmd/set_parameter?layer_enable=on,off,on,off

Reply:
```
{
    "error_code":0,
    "error_text":"success"
}
```

2.7 Alignment aid (pilot laser)

2.7.1 Parameter overview

R2300 devices feature a pilot laser functionality that intended to be used as alignment aid in client applications. This section lists all (global) parameters that are available to configure this feature.

parameter name	type	description	access	default
pilot_laser	bool	enable pilot laser: on / off	vRW	off
pilot_start_angle	int	angle where to enable red pilot laser (1/10 000°)	vRW	-500000
pilot_stop_angle	int	angle where to disable red pilot laser (1/10 000°)	vRW	500000

Please note:
All these parameters are non-persistent, i.e. they revert to their default values after reboot, power cycle or factory reset.

2.7.2 Pilot laser activation (pilot_laser)

The boolean parameter pilot_laser allows to enable (on) or disable (off) the pilot laser functionality. Default is off.

2.7.3 Pilot laser sector (pilot_start_angle and pilot_stop_angle)

The parameters pilot_start_angle and pilot_stop_angle allow to restrict the field of view, where the pilot laser is shown (when enabled via pilot_laser). Per default the parameters define the maximum field of view.

Please note:
The operation of the pilot laser is further restricted by the parameters measure_start_angle / measure_stop_angle (section 2.6.6) and layer_enable (section 2.6.7). The pilot laser will only be visible within the scan acquisition sector of each active scan layer.

2.8 User interface configuration

2.8.1 Parameter overview

This section lists all (global) parameters that are available to configure the sensors human machine interface (HMI).

parameter name	type	description	access	default
locator_indication	bool	LED locator indication: on / off	vRW	off

2.8.2 Locator indication (locator_indication)

The parameter locator_indication temporarily activates a distinctive flashing pattern for the Power and Q2 LEDs. This function can be used to identify a specific R2300 device if multiple devices are installed.

Please note:
The locator indication function is non-persistent, i.e. it is automatically disabled after reboot, power cycle or factory reset.

PEPPERL+FUCHS

2.9 System status

2.9.1 Parameter overview

The following (read only) parameters can be accessed to get status information from the sensor.

parameter name	type	unit	description	access
status information				
`status_flags`	`bitfield`	–	sensor status flags (see section 2.9.2)	RO
time information				
`system_time_raw`	`ntp64`	–	raw system time (see section 3.1.5)	RO
`up_time`	`uint`	min	time since power-on	RO
`power_cycles`	`uint`	–	number of power cycles	RO
`operation_time`	`uint`	min	overall operating time	RO
`operation_time_scaled`	`uint`	min	overall operating time scaled by temperature	RO
operating conditions				
`temperature_current`	`int`	°C	current operating temperature	RO
`temperature_min`	`int`	°C	minimum lifetime operating temperature (power-up update delay 15min)	RO
`temperature_max`	`int`	°C	maximum lifetime operating temperature (power-up update delay 15min)	RO

Example

Query: `http://<sensor IP address>/cmd/get_parameter?list=up_time;power_cycles`

Reply:
```
{
"up_time":44,
"power_cycles":22,
"error_code":0,
"error_text":"success"
}
```


PEPPERL+FUCHS

2.9.2 System status flags (status_flags)

The read-only parameter status_flags (see section 2.9) provides an array of system status flags:

bit	flag name	description
Generic		
0	initialization	System is initializing, valid scan data not available yet
2	scan_output_muted	Scan data output is muted by current system configuration (see section 2.6.2)
3	unstable_rotation	Measured scan rate does not match set value
Warnings		
8	device_warning	Accumulative flag – set if device displays any warning
10	low_temperature_warning	Current internal temperature below warning threshold
11	high_temperature_warning	Current internal temperature above warning threshold
12	device_overload	Overload warning – sensor CPU overload is imminent
Errors		
16	device_error	Accumulative flag – set if device displays any error
18	low_temperature_error	Current internal temperature below error threshold
19	high_temperature_error	Current internal temperature above error threshold
20	device_overload	Overload error – sensor CPU is in overload state
Defects		
30	device_defect	Accumulative flag – set if device detected an unrecoverable defect

System status flags are similar to scan data header status flags (see section 3.4.3) but provide up-to-date information on the current device status (not associated to specific scan data).

Please note:
All flags not listed in the above table are reserved and should be ignored.

PEPPERL+FUCHS

3 Scan data output using UDP

3.1 Principles of scan data acquisition

The R2300 is a *multi-layer laser scanner* designed to periodically measure distances within a 100° field of view for up to 4 distinct measurement planes. The measurements recorded for a single sweep of the measuring beam over the angular field of view are aggregated into a *scan* which yields a sequence of *scan points* (also called *samples*) within this specific scan plane (also called *scan layer*). Scan acquisition is performed at a constant rate as defined by the parameter `scan_frequency` (see section 2.6). The number of scan points within a scan is defined by the parameter `samples_per_scan` (see section 2.6).

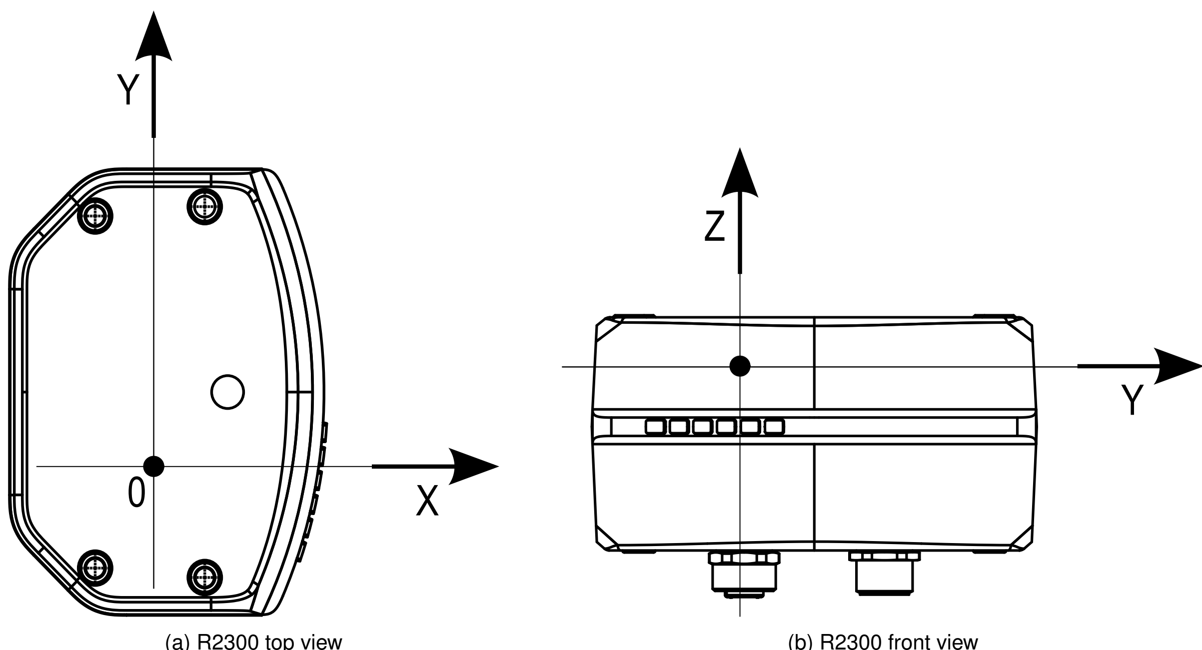
Each scan point is comprised of a distance value for a corresponding angle as well as an echo amplitude. However, since measurements are performed with a uniform angular resolution (depending on the parameter `samples_per_scan`), the actual scan data output typically just gives distance and amplitude data for each sample. The corresponding angular reading can be reconstructed by adding up the angular increments from the starting angle of the scan. The output format of scan data depends on the scan data packet type used – please refer to section 3.4 for further details.

For multi-layer devices successive scans are recorded with different inclination angles (as defined by `layer_inclination`). A set of these scans can be combined to a 3D image *frame* (3D point cloud). The update rate (*frame rate*) of this 3D image is determined by the scan rate (as configured by `scan_frequency`) divided by the number of distinct scan layers (as defined by `layer_count`).

The following subsections describe various basic concepts of the scan data representation used by the R2300.

3.1.1 Sensor coordinate system

The sensor coordinate system is defined as right-handed Cartesian coordinate system. Figure 3.1 shows this coordinate system for the top view and one side view of the sensor: The *origin* is located at the point of intersection of the axis of rotation and the axis of the laser beam. The *X-axis* points to the sensor front (with status LEDs). The *Y-axis* is located perpendicular to the X-axis and parallel to the base-plate of the sensor (pointing upwards in fig. 3.1a). The *Z-axis* is collinear to the axis of rotation (pointing upwards in fig. 3.1b).



(a) R2300 top view

(b) R2300 front view

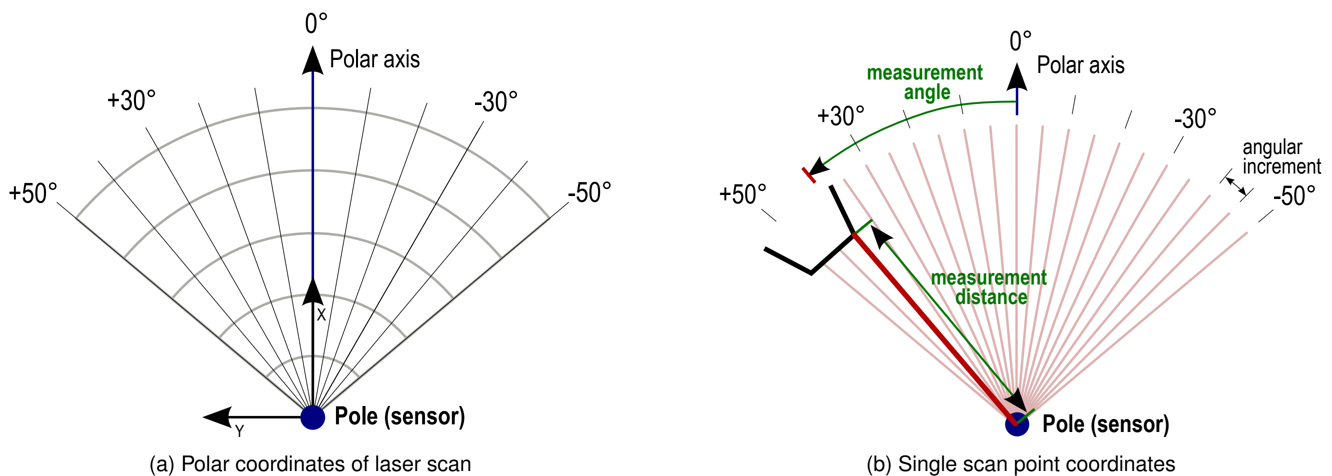
Figure 3.1: Sensor coordinate system

3.1.2 Scan data coordinate system

All measurements of a laser scan are recorded within a specific *scan plane*. R2300devices acquire scan data in up to 4 individual layers which are slightly tilted along the Y-axis compared to the plane formed by the X-axis and the Y-axis of the sensor coordinate system. Therefore each layer defines it's own *scan data coordinate system*.

Scan data acquisition is performed sequentially by a measuring beam rotation around the origin of the scan plane. Therefore scan data is typically represented within a polar coordinate system (see fig. 3.2a). The pole of the polar coordinate system is defined by the axis of rotation (Z-axis of the scan data coordinate system). The reference for angular measurements (polar axis) is equivalent to the X-axis of the scan data coordinate system (pointing upwards in fig. 3.2a).

During nominal operation scan points are continuously recorded using a uniform *angular increment* and direction of rotation. While the angular increment can be configured by a global device parameter (see section 2.6) the direction of rotation is always performed in mathematically positive direction. This direction is called *counter-clockwise* (abbreviated *ccw*) – the angular increment between two subsequent scan points has a positive value.



(a) Polar coordinates of laser scan

(b) Single scan point coordinates

Figure 3.2: Scan data coordinate system

Figure 3.2b shows a (simplified) example of a laser scan with a small number of samples. The *measurement angle* of a single scan point (angular coordinate) is calculated within the scan plane with reference to the polar axis. The measurement distance (radial coordinate) is determined by the distance from the center of rotation (pole) to the object hit by the laser beam. Angular coordinates within the 100° field of view are specified with a value range of $[-50^\circ; +50^\circ]$ including both -50° and $+50^\circ$.

3.1.3 Distance readings

Distance readings are typically output as integer value as defined by the scan data packet type (see section 3.4). In case of invalid measurements (e.g. no echo detected or distance out of range) the distance reading is set to an error substitution value: the biggest representable integer value for a distance value (e.g. 0xFFFFFFFF for an `uint32` typed distance value).

Please note:
The measurement resolution and measurement range are limited by the physical capabilities of the sensor as listed in the sensor data-sheet. This information is also available by means of the read-only variables `radial_resolution`, `radial_range_min` and `radial_range_max` (see section 2.4).

3.1.4 Echo amplitude readings

For each measurement of the sensor optional amplitude data is available to the client. R2300 amplitude data is output as dimensionless non-linear value with a fixed resolution of 12 bit.

On principle, amplitude data can deliver an estimate of the *relative* reflectivity of an object only. Measured amplitude depends on the surface properties of the target object (its *absolute* reflectivity), its distance to the sensor, the angle of incidence of the sensors laser beam on the target surface, etc. – therefore a direct comparison of amplitude data is only viable for object surfaces under similar observation conditions.

Please note:
Please note that amplitude data is not calibrated. Thus amplitude data of different sensor devices may not be identical even under similar observation conditions!

The least significant values of the 12 bit amplitude data are reserved for the following special values:

value	name	description
0	no echo	receiver detected no echo
1	blinding	receiver overloaded due to excessive echo amplitude
2	error	unable to measure echo amplitude
3	reserved	internal (should not occur during normal operation)
4	reserved	internal (should not occur during normal operation)
5	reserved	internal (should not occur during normal operation)
6	weak echo	detected echo too weak for a valid measurement
7-31	reserved	reserved for internal use
>31	amplitude	measured echo amplitude value

All values in the range of 7 to 31 are reserved for internal use. The smallest amplitude value for a *valid* measurement is 32.

3.1.5 Timestamps

The R2300 devices provide *raw* timestamps in 64bit NTP timestamp format (see section 2.1.8 for details). They are generated by an internal system clock that starts counting from zero at power-on. Its resolution is better than 1 ms and its drift is below 100 ppm. Raw time is always incrementing without any discontinuities or overflows.

The raw system time can be accessed via the device parameter `system_time_raw` (see section 2.9). When `system_time_raw` is read using `get_parameter` the device will return the raw system time for the point in time, when the command has been received. Please note that both sending the request for a timestamp and receiving the reply with the timestamp are affected by the non-deterministic HTTP transmission delay.

3.2 Principles of scan data output

3.2.1 Introduction

In order to receive scan data from the laser scanner the client application needs to establish a *scan data connection* to the sensor. R2300 devices currently support UDP data channels only. UDP channels allow data transmission with low latency at the expense of potential unrecoverable data corruption or data loss. They are managed using the HTTP command interface.

For typical applications the following steps are necessary to use scan data output:

1. Set up global configuration of the scanner (see chapter 2), if necessary
2. Establish a data channel to the sensor (see section 3.3.1)
3. Configure scan data output (see section 3.3.5), if necessary
4. Start scan data transmission (see section 3.3.3)
5. Receive scan data from the device (see section 3.4)
6. Stop scan data transmission (see section 3.3.4)
7. Terminate the data channel to the sensor (see section 3.3.2)

Section 3.3 covers the required commands for managing scan data output in detail.

3.2.2 Scan data connection handles

The PFSDP protocol supports parallel scan data connections to multiple clients. In order to configure and control these connections individually, each connection is identified by a unique connection *handle*. A handle is defined as random alphanumeric string of maximal 16 characters. The sensor ensures that each handle is used for only one active scan data connection. Applications should not make any further assumption regarding the structure of a handle as implementation details might change with new firmware versions (see also below).

Please note:
Due to limited system ressources R2300 devices support a *single* scan data connection only. Nevertheless the PFSDP handle mechanism is used to manage that connection.

3.2.3 Scan data output customization

A client may customize some properties of how scan data is output over a scan data channel. These configuration settings are specific to a single data connection identified by the unique connection handle. The settings can be set while initiating a new scan data connection using `request_handle_udp`(see section 3.3.1), or can be changed for an existing scan data connection using `set_scanoutput_config` (see section 3.3.5).

Selecting a start angle

A client may define a (virtual) start angle for scan data output using the parameter `start_angle` (see section 3.3.5). All scan points recorded *before* this start angle (in scan direction) are discarded. The first scan point (index 0) of a scan has a angle which is equal to or behind the given start angle. The parameter `start_angle` does not control the angular value at which scan points are *recorded*. It only defines a criterion, which scan points should be *output* for a specific scan data connection.

The value of `start_angle` refers to the (polar) measurement angle of the scan data coordinate system (see section 3.1.2). By default `start_angle` is set to the beginning of the sensors angular field of view at $-50.0°$ (i.e. `start_angle` $= -500000$). Subsequent scan points (index $(n+1)$) within the scan data stream are ordered according to the direction of rotation of the measuring beam.

Please note:
The sensors angular field of view can be further limited using the parameters `measure_start_angle` and `measure_stop_angle` (see section 2.6.6). The value of `start_angle` can be configured outside these limits, but the resulting scan output will always contain values from the angular measuring range only.

PEPPERL+FUCHS

Limiting the number of scan points

The parameter `max_num_points_scan` allows to limit the number of scan points that are *output* over a scan data connection. In contrast to the global parameter `samples_per_scan`, which reflects how many samples per scan are *recorded* by the sensor, the setting of `max_num_points_scan` affects the number of scan points *output* for a specific scan data connection only.

If `max_num_points_scan` is set to a value below `samples_per_scan`, the client application receives less scan points than the sensor records. In combination with the parameter `start_angle` this allows client application to obtain only a segment (sector) of a scan instead of all recorded scan points. Figure 3.3 visualizes such a setup. This can be very useful to reduce data traffic if the full field of view of the sensor is not needed.

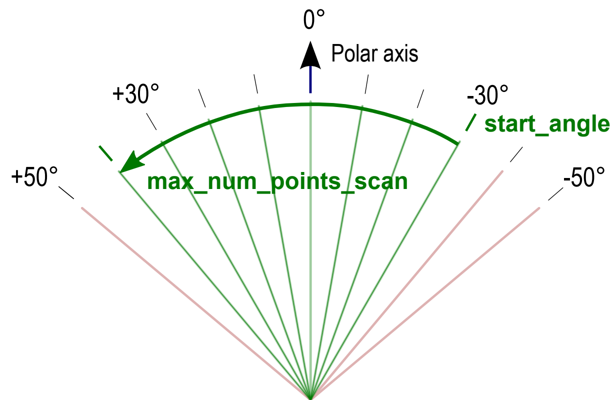


Figure 3.3: Restriction of scan output to a segment

> **Please note:**
> Setting `max_num_points_scan` to a value above `samples_per_scan` will have no effect. The resulting scan output will not contain any additional (dummy) samples.

3.3 Commands for managing scan data output

The subsequent sections describe all commands.

3.3.1 `request_handle_udp` – request for an UDP-based scan data channel

The command `request_handle_udp` is used to request a handle for an UDP-based scan data transmission from the sensor to the client. If successful the sensor will send scan data to the client using the target IP address and UDP port specified at the handle request. Figure 3.4 gives an overview on the communication between sensor and client when using an UDP-based channel for scan data output.

Command arguments

The command `request_handle_udp` accepts the following arguments:

argument name	type	unit	description	default
address	ipv4	–	required: target IP address of the client	–
port	uint	–	required: target port for UDP data channel (client side)	–
packet_type	enum	–	optional: scan data packet type: C1 (see section 3.4)	C1
start_angle	int	1/10 000°	optional: angle of first scan point for scan data output	-500000
max_num_points_scan	uint	samples	optional: limit number of points in scan data output	0 (unlimited)

The optional arguments of `request_handle_udp` facilitate an adequate initial configuration of the scan data output, which can be later modified using the command `set_scanoutput_config`. Please refer to section 3.3.5 for a detailed description of these optional arguments.

Command return values

- `handle` – unique (random) alpha-numeric string as identifier (handle) for the new UDP data channel

During a valid command call the scanner creates a new UDP channel to the client using the specified target IP address and port number. In case of an error the returned value for `handle` is invalid and `error_code` / `error_text` return details regarding the cause of the negative response (see section 1.2.7).

Please note:
When a new UDP connection handle is requested the scanner will close any previously requested UDP connection. R2300 devices support a single scan data connection only (as denoted by `max_connections`, see section 2.4.1).

Please note:
Since an UDP scan data connection is established from the sensor to the client ("incoming connection") it is prone to be blocked by firewall software. Please ensure that your firewall settings allow incoming UDP connections from the sensor IP address to your client application.

Please note:
Applications should not make any assumption regarding the structure of a handle. Handles should be treated as random alpha-numeric string of max. 16 characters.

Command example

Query: `http://<sensor IP address>/cmd/request_handle_udp?address=192.168.10.20&port=54321&packet_type=C`

Reply:
```
{
    "handle":"s10",
    "error_code":0,
    "error_text":"success"
}
```

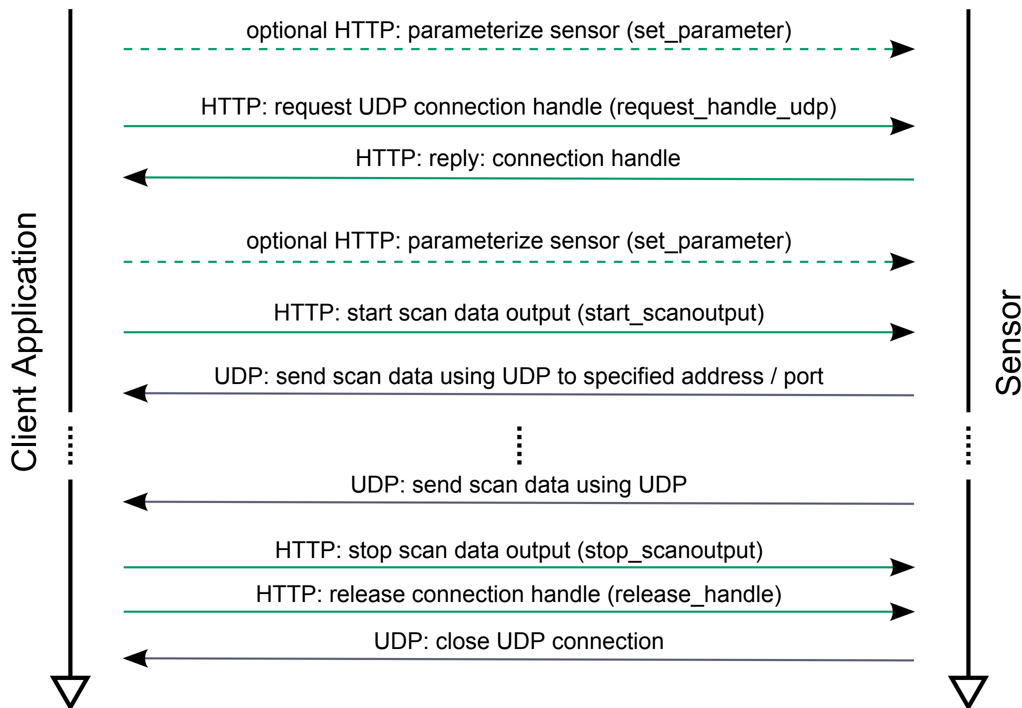


Figure 3.4: Timeline: scan data transmission using UDP

3.3.2 `release_handle` – release a data channel handle

Using the command `release_handle` the client can release a data channel handle. Any active scan data output using this handle will be stopped immediately. An associated UDP-based data channel is closed by the sensor itself. An associated TCP-based data channel should be closed by the client.

Command arguments

argument name	type	description
`handle`	`string`	handle for scan data channel (max. 16 chars) (required argument – always specified first)

Command example

Query: `http://<sensor IP address>/cmd/release_handle?handle=s22`

Reply:
```
{
"error_code":0,
"error_text":"success"
}
```

3.3.3 `start_scanoutput` – initiate output of scan data

The command `start_scanoutput` starts the transmission of scan data for the data channel specified by the given handle. When started, the sensor will begin sending scan data to the client using an established UDP or TCP channel with the given handle – see section 3.3.1. (Re-)starting a scan data transmission also resets the counters for scan number and scan packet number in the scan data header (see section 3.4.2). Scan data output always starts at the beginning of a new scan (with scan number 0 and scan packet number 1).

Protocol version 1.05 Document release 2022-08

PEPPERL+FUCHS

Command arguments

argument name	type	description
`handle`	`string`	handle for scan data channel (max.16 chars) (required argument – always specified first)

Command example

Query: `http://<sensor IP address>/cmd/start_scanoutput?handle=s22`

Reply:
```
{
"error_code":0,
"error_text":"success"
}
```

3.3.4 `stop_scanoutput` – terminate output of scan data

The command `stop_scanoutput` stops the transmission of scan data for the data channel specified by the given handle. Scan data output stops immediately after the current scan data packet – not necessarily at the end of a full scan.

Please note:
TCP clients might still receive several scan data packets after sending `stop_scanoutput`, due to the TCP stack data queue.

Command arguments

argument name	type	description
`handle`	`string`	handle for scan data channel (max.16 chars) (required argument – always specified first)

Command example

Query: `http://<sensor IP address>/cmd/stop_scanoutput?handle=s22`

Reply:
```
{
"error_code":0,
"error_text":"success"
}
```

3.3.5 `set_scanoutput_config` – reconfigure scan data output

Using the command `set_scanoutput_config` the client can parametrize scan data output separately for each active scan data output channel. All command arguments solely apply to the *output* of scan data. Customization of (global) parameters referring to the recording of measurements (scan data) is done by use of the command `set_parameter` (see section 2.6).

Command arguments

argument name	type	unit	description	default
`handle`	`string`	–	handle for scan data channel (max.16 chars) (required argument – always specified first)	–
`packet_type`	`enum`	–	optional: scan data packet type: `C1` (see section 3.4)	`C1`
`start_angle`	`int`	1/10 000°	optional: angle of first scan point for scan data output	`-500000`
`max_num_points_scan`	`uint`	samples	optional: limit number of points in scan data output	`0` (unlimited)

It is recommended (but not required) to stop sensor data output while using `set_scanoutput_config`. In case scan data output is active, the point in time when modified configuration settings are applied to the running data stream is non-deterministic. After the new settings are applied, scan data output is suspended until the start of an new scan (skipping scan data packets in-between). If the client application depends on a deterministic switching behavior, it should stop scan data transmission first using `stop_scanoutput`, change settings using `set_scanoutput_config` and finally restart the data stream with `start_scanoutput`.

Parameter `start_angle`

The user can control the angle for the first scan point of a scan by means of the parameter `start_angle`. The range of valid values is [`-1800000`; `+1800000`[including `-1800000` (−180°) but excluding `+1800000` (+180°). The specified value does not have to match the configured angular resolution for scan data acquisition (see section 2.6) – the sensor will start scan data output with the first scan point whose recording angle is equal to or following behind the specified angle in direction of rotation.

> **Please note:**
> The command `get_scanoutput_config` (see section 3.3.6) will return the exact user specified value, while the entry “absolute angle of first scan point” within the scan data packet header (see section 3.4.2) will specify the exact value of the first scan point actually used.

Command example

Query: `http://<sensor IP address>/cmd/set_scanoutput_config?handle=s22&packet_type=B&start_angle=-900000`

Reply:
```
{
"error_code":0,
"error_text":"success"
}
```

Parameter `max_num_points_scan`

This parameter allows to limit the number of samples that are output for each scan. In combination with the parameter `start_angle` a client application can reduce scan data output to a single region of interest (*sector*). Please refer to section 3.2.3 for further details.

The parameter is specified as unsigned integer (`uint`) and accepts any non-negative number. The value `0` is recognized as special case for 'no limitation', i.e. the sensor outputs always all points of scan. This is also the default value.

3.3.6 `get_scanoutput_config` – read scan data output configuration

The command `get_scanoutput_config` returns the current scan data output configuration for a specified scan data output channel (UDP or TCP).

Command arguments

argument name	type	description
handle	string	handle for scan data channel (max.16 chars) (required argument – always specified first)
list	string	semicolon separated list of parameter names (optional)

If the argument `list` is not specified the command will return the current value of all available configuration parameters (see section 3.3.5).

Command example

Query: `http://<sensor IP address>/cmd/get_scanoutput_config?handle=s22`

Reply:

```
{
"address":"0.0.0.0",
"port":39050,
"watchdog":"on",
"watchdogtimeout":60000,
"packet_type":"A",
"start_angle":-1800000,
"error_code":0,
"error_text":"success"
}
```

3.4 Transmission of scan data

Scan data is always transmitted within packets. A complete scan is usually transmitted using multiple scan data packets (see section 1.1 for basic design considerations). Each packet comprises of a generic header, a scan data specific header and the actual scan data.

A new scan will always start with a new scan data packet, i.e. the first sample of a new scan will always appear as first sample of a new packet. Each scan data packet is transmitted as soon as the required data is available. This *streaming* approach allows a client application to start processing scan data with minimal delay – eliminating the need to wait until the full scan is recorded and transmitted to the client completely.

Multiple scan data packet types are defined to output different sets of scan data information efficiently. These packet types follow a standard structure – differing in the bulk scan data only. Within bulk scan data each scan point is represented by a structure containing the favored amount of data (distance, amplitude, etc.). The following sections describe scan data packets in detail.

3.4.1 Basic packet structure

Each data packet has the following basic structure:

type	name	description
`uint16`	`magic`	magic byte (0xa25c) marking the beginning of a packet
`uint16`	`packet_type`	type of scan data packet (low-byte: payload type, high-byte: header type)
`uint32`	`packet_size`	overall size of this packet in bytes (header, payload, checksum)
`uint16`	`header_size`	size of header in bytes (i.e. offset to payload data)
...	...	packet type specific additional header information
`uint8[]`	`header_padding`	0-3 bytes padding (to align the header size to a 32bit boundary)
...	`payload_data`	packet type specific payload data
`uint8[]`	`payload_padding`	0-3 bytes padding (to align the payload size to a 32bit boundary)

Please note:
Although the structure of the packet usually appears to be fixed, it is highly recommended that client applications always evaluate the entries for packet size and header size since they may change due to future extensions.

The magic byte at the beginning of the packet header is designed to be used as synchronization mark within a continuous data stream. It can be ignored if synchronization is not needed.

The starting address of payload data is always aligned to a 32bit address boundary by using padding bytes within the header (`header_padding`). Additionally, the overall size of the packet is always aligned to 32bit boundary. Depending on the scan data packet type there might be additional padding bytes (`payload_padding`) at the end of the packet.

3.4.2 Typical structure of a scan data header

A scan data packet contains a scan data header with information on the scan and the scan data itself. The scan data header is designed in ways that each scan data packet can be processed independent of other scan data packets belonging to the same scan.

A typical scan data header has the following structure:

type	name	description
uint16	magic	magic byte (0xa25c) marking the beginning of a packet
uint16	packet_type	type of scan data packet
uint32	packet_size	overall size of this packet in bytes (header, payload, checksum)
uint16	header_size	size of header in bytes (i.e. offset to payload data)
uint16	scan_number	sequence number for scan (counting transmitted scans, starting with 0, overflows)
uint16	packet_number	sequence number for packet (counting packets of a particular scan, starting with 1)
uint16	layer_index	index of scan layer (starting with 0)
int32	layer_inclination	angle of scan layer inclination ($1/10\,000°$)
ntp64	timestamp_raw	raw timestamp of first scan point in this packet (see section 3.1.5)
uint64	reserved	reserved field
uint32	status_flags	scan status flags (see section 3.4.3)
uint32	scan_frequency	currently configured scan rate ($1/1000$ Hz)
uint16	num_points_scan	number of scan points (samples) within complete scan (depending on configured FOV)
uint16	num_points_packet	number of scan points within this packet
uint16	first_index	index of first scan point within this packet
int32	first_angle	absolute angle of first scan point in this packet ($1/10\,000°$)
int32	angular_increment	delta angle between two scan points ($1/10\,000°$) (CCW rotation: positive increment, CW rotation: negative increment)
uint32	reserved	reserved field
uint32	reserved	reserved field
uint64	reserved	reserved field
uint64	reserved	reserved field
uint8[]	header_padding	0-3 bytes padding (to align the header size to a 32bit boundary)
...	scandata	packet type specific scan data
uint8[]	payload_padding	0-3 bytes padding (to align the payload size to a 32bit boundary)

Please note:
The field `num_points_scan` states the total number of scan points *output* for each recorded scan. It is always equal to either `samples_per_scan` or `max_num_points_scan`, whichever is smaller for the specific scan data connection. Please refer to section 3.2.3 for more details on this matter.

Please note:
The field `scan_frequency` states the current *target value* for the scan acquisition rate as defined by the global parameter `scan_frequency` (see section 2.6.3). The *instantaneous value* of the angular velocity of the measuring beam can be estimated from the timestamps of the current and subsequent scan data packets.

Please note:
Angular values specified with a resolution of $1/10\,000°$ are usually prone to rounding errors due to the decimal range of values. They are part of the header for convenience only. Subsequent calculations requiring precise angular values should calculate an exact angle for each scan point by reference to its index number, the configured angular increment and the configured start angle of the scan:

CCW rotation: $$exact_angle_{scanpoint} = start_angle_{scan} + index_{scanpoint} * \frac{angular_fov}{num_points_scan}$$

CW rotation: $$exact_angle_{scanpoint} = start_angle_{scan} - index_{scanpoint} * \frac{angular_fov}{num_points_scan}$$

PEPPERL+FUCHS

3.4.3 Scan data header status flags

Scan data header status flags are similar to system status flags (see section 2.9.2) but provide status information specific to the scan data of a scan data packet. Each scan data header contains an `uint32` entry `status_flags` (see section 3.4.2) comprised of the following flags:

bit	flag name	description
Informational		
0	`scan_data_info`	Accumulative flag – set if any informational flag (bits `1..7`) is set
1	`new_settings`	System settings for scan data acquisition changed during recording of this packet. This flag is triggered by write accesses to global parameters affecting the measuring configuration (see section 2.6) which can be done by any client. Changes to connection-specific parameters (see section 3.3.5) do not trigger this flag!
2	`invalid_data`	Consistency of scan data is not guaranteed for this packet.
3	`unstable_rotation`	Measured scan rate did not match set value while recording this scan data packet.
4	`skipped_packets`	Preceding scan data packets have been skipped due to connection issues, changes to scan data acquisition settings or scan data inconsistencies.
Warnings		
8	`device_warning`	Accumulative flag – set if any warning flag (bits `9..15`) is set
10	`low_temperature_warning`	Current internal temperature below warning threshold
11	`high_temperature_warning`	Current internal temperature above warning threshold
12	`device_overload`	Overload warning – sensor CPU overload is imminent
Errors		
16	`device_error`	Accumulative flag – set if any error flag (bits `17..23`) is set
18	`low_temperature_error`	Current internal temperature below error threshold
19	`high_temperature_error`	Current internal temperature above error threshold
20	`device_overload`	Overload error – sensor CPU is in overload state
Defects		
30	`device_defect`	Accumulative flag – set if device detected an unrecoverable defect

Please note:
All flags not listed in the above table are reserved and should be ignored.

3.4.4 Scan data packet type C1 – distance and amplitude

Scan data packets of type C1 have the following structure:

type	name	description
packet header		
uint16	magic	magic byte (0xa25c) marking the beginning of a packet
uint16	packet_type	type of scan data packet: 0x3143 (ASCII characters 'C1')
uint32	packet_size	overall size of this packet in bytes (header, payload, checksum)
uint16	header_size	size of header in bytes (i.e. offset to payload data)
uint16	scan_number	sequence number for scan (counting transmitted scans, starting with 0, overflows)
uint16	packet_number	sequence number for packet (counting packets of a particular scan, starting with 1)
uint16	layer_index	index of scan layer (starting with 0)
int32	layer_inclination	angle of scan layer inclination ($1/10\,000°$)
ntp64	timestamp_raw	raw timestamp of first scan point in this packet (see section 3.1.5)
uint64	reserved	reserved field
uint32	status_flags	scan status flags (see section 3.4.3)
uint32	scan_frequency	currently configured scan rate ($1/1000$ Hz)
uint16	num_points_scan	number of scan points (samples) within complete scan (depending on configured FOV)
uint16	num_points_packet	number of scan points within this packet
uint16	first_index	index of first scan point within this packet
int32	first_angle	absolute angle of first scan point in this packet ($1/10\,000°$)
int32	angular_increment	delta angle between two scan points ($1/10\,000°$) (CCW rotation: positive increment)
uint32	reserved	reserved field
uint32	reserved	reserved field
uint64	reserved	reserved field
uint64	reserved	reserved field
uint8[]	header_padding	0-3 bytes padding (to align the header size to a 32bit boundary)
scan point data		
uint20	distance	measured distance (in mm) – maximum representable value is 1 km Invalid measurements return 0xFFFFF.
uint12	amplitude	measured amplitude Please see section 3.1.4 for a description of amplitude data values.
	...	

Please note:
The field num_points_scan states the total number of scan points *output* for each recorded scan. It is always equal to either samples_per_scan or max_num_points_scan, whichever is smaller for the specific scan data connection. Please refer to section 3.2.3 for more details on this matter.

Please note:
The field scan_frequency states the current *target value* for the scan acquisition rate as defined by the global parameter scan_frequency (see section 2.6.3). The *instantaneous value* of the angular velocity of the measuring beam can be estimated from the timestamps of the current and subsequent scan data packets.

3.5 Data transmission using UDP

The UDP/IP-based scan data output provides a low latency channel for scan data transmission. Each scan data packet is send as separate UDP message (datagram) using (at least) one Ethernet frame. In case an UDP message (scan data packet) is lost during transmission, no error correction is provided. Corrupted scan data packets are discarded. The client application can make use of all successfully received scan data packets though, since every scan data packet incorporates a full scan data header which allows to process the contained scan data separately.

4 Filter-based scan data processing

4.1 Introduction to scan data filtering

The R2300 devices optionally provide support for in-device pre-processing of scan data. This gives customer applications the options to either reduce the amount of received data while utilizing the full scan resolution with *block-wise* processing or to just apply a filtering algorithm to all scan data with *moving-window* processing.

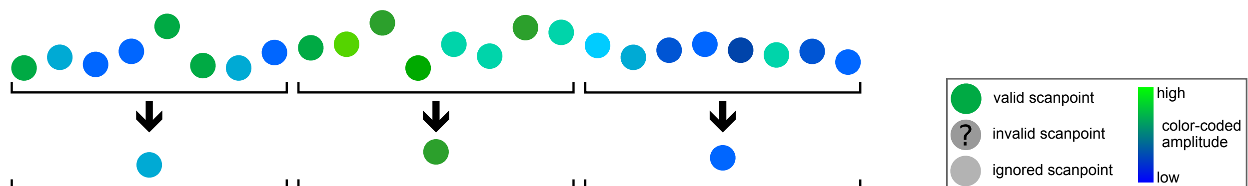
The basic idea of scan data filtering is to combine a configurable number of N adjacent scan points (input values) into a single resulting scan point (output value) using one of various predefined algorithms. A filter algorithm calculates both a distance value and an amplitude value from the input data. The resulting scan point is placed at the center of the processing window for both angular value and timestamp value. All operations are performed in the sensor coordinate system (see section 3.1.1).

Please note:
Scan data filtering is applied *globally*, i.e. its settings affect *all clients*. It should be treated similar to the (global) measuring configuration (see section 2.6).

Scan data filtering can be considered as transparent to a client application. On protocol level there is no difference between a scan recorded with a lower resolution and a scan recorded with a high resolution and scan data filtering enabled. However, the latter provides a potentially higher signal quality.

4.1.1 Block-wise processing

Filter algorithms with block-wise processing calculate a single output value for N input values. After processing the input values the input window is shifted by N values, i.e. each input value is processed only once. Thus the number of points in the resulting scan is reduced by a *decimation* factor of $1 : N$ compared to the input scan, i.e. the output scan contains only $1/N$ scan points and has a N times coarser scan resolution (with a constant angular increment).

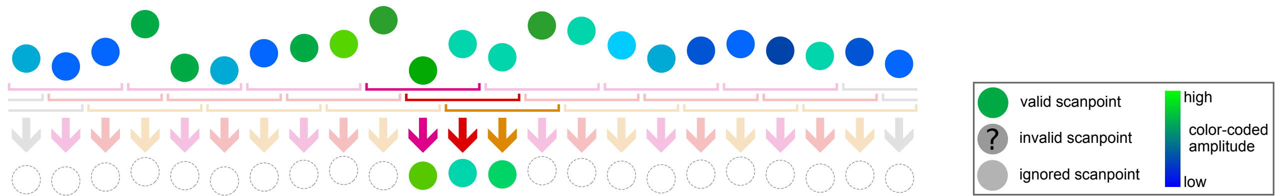


Above figure shows an example for the decimation process of 24 input scan points with a window size of 8 points. Each scan point is represented by a circle with a color-encoded echo amplitude (blue: low echo, green: high echo). The filtered result contains only 3 output scan points – on for each group of input values (8:1 decimation).

PFSDP compatibility note:
Block-wise scan data filtering requires a device with PFSDP version 1.03 or newer. Furthermore the device must support the device feature `scan_data_filter` – please refer to section 2.4 for details on sensor capabilities.

4.1.2 Moving-window processing

Filter algorithms with moving-window processing calculate a single output value from N input values. After processing the input values the input window is shifted by 1, i.e. each input value is incorporated into N consecutive output values. The resolution of the resulting scan is equal to the resolution of the original input scan (no decimation).



Above figure shows an example for the filtering process for a continuous stream of input scan points. Again each scan point is represented by a circle with a color-encoded echo amplitude (blue: low echo, green: high echo). The moving-window uses a window size of 3 points here. The filtered result contains a similar amount of output scan points – one for each group of 3 input values.

PFSDP compatibility note:
Moving-window scan data filtering requires a device with PFSDP version 1.05 or newer. Furthermore the device must support the device feature `scan_data_filter_moving` – please refer to section 2.4 for details on sensor capabilities.

4.1.3 Filter processing at scan edges

The global parameters `measure_start_angle` and `measure_stop_angle` determine the scan data acquisition sector (see section 2.6.6), i.e. the angular field of view where measurements are recorded as input for the scan data filter. Since the R2300 does not feature a full 360° field of view, the filter processing needs to pay particular attention at the edges of a scan (first and last scan points).

For both block-wise and moving-window processing the first filtered sample appears at the center of the first processing window which begins directly after `measure_start_angle`. The last filtered sample appears at the center of the last processing window which ends directly before `measure_stop_angle`. Consequently the first $(\texttt{filter_width} - 1)/2)$ input scan points and the last $(\texttt{filter_width}-1)/2)$ input scan points have no direct counterpart in the output scan.

The following figure shows an example for a moving-window filter with window size 5 at the start and at the end of a scan:

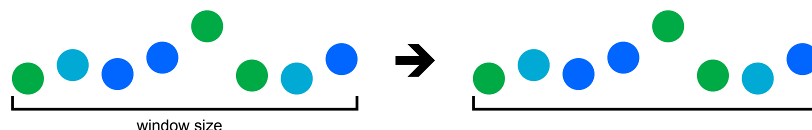
Please note:
The number of *filtered* scan points that are output via a scan data connection can be further reduced using the connection-specific parameters `start_angle` and `max_num_points_scan`. Please refer to section 3.3.5 for details.

4.2 Filter algorithms

This section describes the available algorithms for scan data filtering, selectable by the global parameter `filter_type`. All parameters are discussed in detail in section 4.3.

4.2.1 No filter (pass-through)

Per default no filtering is performed on sensor data. All recorded scan points are passed-through to the client without change. This behavior is identical to devices that do not support scan data filtering (e.g. older firmware releases).

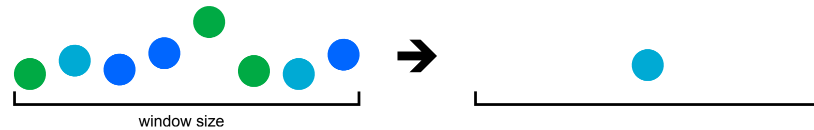


Related configuration parameters: –

4.2.2 Average filter

The *average* filter calculates a simple arithmetic average (distance and amplitude) of all scan data points within the configured window size (`filter_width`).

For block-wise processing the result is a single output scan point replacing the complete group of input scan points:



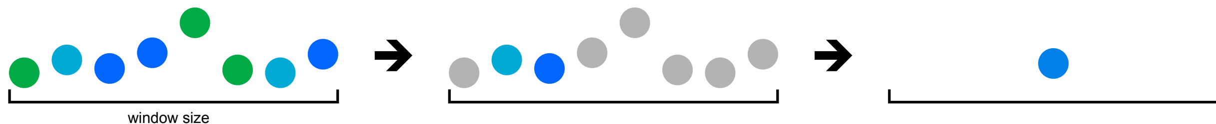
For moving-window processing the filter works similar but a filtered output sample is calculated for each input sample (with the configured window size).

Related configuration parameters: `filter_width`, `filter_error_handling`

4.2.3 Median filter

The *median* filter calculates a median value from all scan data points within the configured window size (`filter_width`). For this purpose, first all scan points are (virtually) sorted by their distance value. For *odd* window sizes the middle scan point is selected as output sample. For *even* window sizes the two middle points are selected and the output sample is calculated as arithmetical average (for both distance and amplitude) of these points.

For block-wise processing the resulting scan point replaces the group of input scan points:



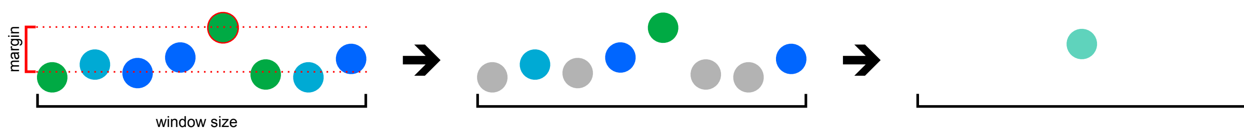
For moving-window processing the filter works similar but a filtered output sample is calculated for each input sample (with the configured window size).

Related configuration parameters: `filter_width`, `filter_error_handling`

4.2.4 Maximum filter

The *maximum* filter is a more complex filter operation. It calculates the arithmetic average from a subset of scan points within the configured filter window (`filter_width`). Scan points are selected by first determining the scan point with the maximum distance within the current filter window. Then all scan points within this window are eliminated, whose distance value falls below the maximum distance value less a threshold value (`filter_maximum_margin`). The remaining points are used to calculate an arithmetic average for both distance and amplitude.

For block-wise processing the resulting scan point replaces the group of input scan points:



For moving-window processing the filter works similar but a filtered output sample is calculated for each input sample (with the configured window size).

Related configuration parameters: `filter_width`, `filter_error_handling`, `filter_maximum_margin`

4.3 Filter configuration

Scan data filtering is configured globally using the commands for sensor parametrization (see section 2.2). This section gives an overview on the available settings.

4.3.1 Parameter overview

The following (global) parameters are available for configuration of scan data filtering:

parameter name	type	unit	description	access	default
`filter_type`	`enum`	–	algorithm for filtering (see section 4.3.2 for details)	RW	`none`
`filter_width`	`uint`	samples	window size for filtering (see section 4.3.3 for details)	RW	`4`
`filter_error_handling`	`enum`	–	strategy for filtering invalid values (see section 4.3.4 for details)	RW	`tolerant`
`filter_maximum_margin`	`uint`	1 mm	margin for filter type `maximum` (see section 4.3.5 for details)	RW	100 mm

4.3.2 Filter types (`filter_type`)

The parameter `filter_type` selects the filtering algorithm that is applied globally to all scan data recorded by the sensor. Currently, the following algorithms are available (see section 4.2 for details):

filter type	description
`none`	Filtering disabled. Output all recorded samples (pass through).
`average`	Calculate arithmetic average from N raw samples (see section 4.2.2).
`median`	Calculate median from N raw samples (see section 4.2.3).
`maximum`	Filter raw samples by distance and calculate average (see section 4.2.4).

Example

Query: `http://<sensor IP address>/cmd/set_parameter?filter_type=average`

Reply:
```
{
"error_code":0,
"error_text":"success"
}
```

4.3.3 Filter width (`filter_width`)

The parameter `filter_width` controls the window size of the filter algorithm applied to all recorded scan data. It defines the number of recorded samples (scan data points) that are processed to produce a (single) filtered output sample.

R2300 devices currently support the following window sizes: `2`, `3`, `4`, `5`, `7`, `8`, `15`, `16`

> **Please note:**
> The filtered output scan point is always placed at the center of the filter window for both angular value and timestamp value (see section 4.1).

Example

Query: `http://<sensor IP address>/cmd/set_parameter?filter_width=4`

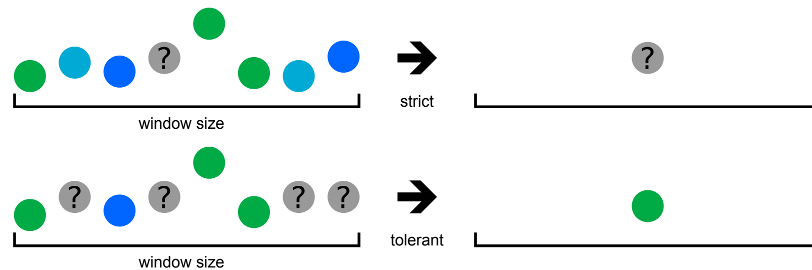
Reply:
```
{
"error_code":0,
"error_text":"success"
}
```

4.3.4 Filter error handling (`filter_error_handling`)

The parameter `filter_error_handling` specifies how the filter algorithm is handling invalid measurement values within the group of scan data points as configured by `filter_width`.

parameter value	description
`strict`	Result is invalid, if any scan data point of the group is invalid.
`tolerant`	Result is valid, if at least one scan data point of the group is valid.

The following pictures illustrate this behavior:



Example

Query: `http://<sensor IP address>/cmd/set_parameter?filter_error_handling=tolerant`

Reply:
```
{
    "error_code":0,
    "error_text":"success"
}
```

4.3.5 Maximum filter margin (`filter_maximum_margin`)

The parameter `filter_maximum_margin` is evaluated by the *maximum filter* algorithm (see section 4.2.4). It defines the allowed distance of a scan point to the maximum distance value within the group of scan data points. The parameter has a resolution of 1 mm and accepts values in the range from 0 mm up to 65 535 mm.

Example

Query: `http://<sensor IP address>/cmd/set_parameter?filter_maximum_margin=220`

Reply:
```
{
    "error_code":0,
    "error_text":"success"
}
```


PEPPERL+FUCHS

5 Advanced topics

This chapter covers various advanced topics about using R2300 devices in more complex applications.

5.1 Device discovery using SSDP

The R2300 provides support for the *Simple Service Discovery Protocol (SSDP)* [15] in order to discover any R2300 devices and their associated IP address within the Ethernet network. SSDP uses UDP multicast messages to query SSDP aware devices.

In order to discover all R2300 devices, the following steps need to be performed:

1. Send a SSDP search request.
2. Process SSDP replies from devices.
3. Read a SSDP device description from each device for additional information.

The following sections describe each step in detail.

5.1.1 SSDP search request

The first step of the SSDP device discovery is to issue a search request on the local network. For this purpose an UDP listener needs to be opened on the local UDP port 1900. Then an UDP datagram with the following content needs to be sent to the UDP multicast address 239.255.255.250 at port 1900:

```
M-SEARCH * HTTP/1.1
HOST: 239.255.255.250:1900
ST: urn:pepperl-fuchs-com:device:R2300:1
MAN: "ssdp:discover"
MX: 1
```

The specified URN addresses R2300 devices only. Other SSDP aware devices on the network will ignore this request.

Please note:
On a client PC with multiple network adapters, the SSDP search request needs to be performed on each network adapter.

5.1.2 SSDP device reply

The second step of the discovery procedure requires the client application to wait for replies to the above search request using the created UDP listener. Each R2300 device on the local network will answer the search request with a message similar to this example:

```
HTTP/1.1 200 OK
LOCATION: http://10.0.10.76/ssdp.xml
SERVER: urhttpd/1.0 UPnP/1.0 R2300/1.0
CACHE-CONTROL: max-age=1800
EXT:
ST: urn:pepperl-fuchs-com:device:R2300:1
USN: uuid:7df9a5ed-07f6-45e1-ac55-3335a4057a10::urn:pepperl-fuchs-com:device:R2300:1
```

This reply contains two important pieces of information:

- The line `LOCATION:` contains the IP address of the device within an URL pointing to a more detailed SSDP device description (see next section).
- The line `USN:` contains an unique identifier (uuid) for this specific device. This uuid allows to identify this R2300 device even if its IP address changes.

5.1.3 SSDP device description

The final step of the SSDP discovery procedure is to obtain the XML based device description. This step can be skipped, if no detailed information on the discovered devices are needed. R2300 devices provide a `ssdp.xml` file at the URL from the `LOCATION` field of the SSDP device reply (see previous section):

```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
  <specVersion>
    <major>1</major>
    <minor>0</minor>
  </specVersion>
  <device>
    <deviceType>urn:pepperl-fuchs-com:device:R2300:1</deviceType>
    <friendlyName>OMD10M-R2300-B23-V1V1D-4S (#40000069736491)</friendlyName>
    <modelDescription>2-D LiDAR sensor</modelDescription>
    <modelName>OMD10M-R2300-B23-V1V1D-4S</modelName>
    <modelNumber>322226</modelNumber>
    <serialNumber>40000069736491</serialNumber>
    <UDN>uuid:7df9a5ed-07f6-45e1-ac55-3335a4057a10</UDN>
    <manufacturer>Pepperl+Fuchs</manufacturer>
    <manufacturerURL>http://www.pepperl-fuchs.com</manufacturerURL>
    <serviceList>
      <service>
        <serviceType>urn:pepperl-fuchs-com:service:none:1</serviceType>
        <serviceId>urn:pepperl-fuchs-com:serviceId:none</serviceId>
        <controlURL>/cmd/</controlURL>
        <eventSubURL></eventSubURL>
        <SCPDURL>/service.xml</SCPDURL>
      </service>
    </serviceList>
    <iconList>
      <icon>
        <id>0</id>
        <mimetype>image/png</mimetype>
        <width>48</width>
        <height>48</height>
        <depth>24</depth>
        <url>/device.png</url>
      </icon>
    </iconList>
    <X_pfsdpVersionMajor>0</X_pfsdpVersionMajor>
    <X_pfsdpVersionMinor>5</X_pfsdpVersionMinor>
    <X_pfsdpDeviceFamily>5</X_pfsdpDeviceFamily>
    <X_dtmAccessPath>/dtm/</X_dtmAccessPath>
  </device>
</root>
c@Fancy
```

The standard SSDP XML device description contains already various useful fields:

- `manufacturer` – vendor name of the device (see parameter `vendor` in section 2.3)
- `modelName` – product name of the device (see parameter `product` in section 2.3)
- `modelNumber` – part number of the device (see parameter `part` in section 2.3)
- `serialNumber` – serial number of the device (see parameter `serial` in section 2.3)

R2300 devices additionally provide the following non-standard items with PFSDP specific information:

- `X_pfsdpVersionMajor` – major PFSDP protocol revision (see `version_major` in section 1.2.8)
- `X_pfsdpVersionMinor` – minor PFSDP protocol revision (see `version_minor` in section 1.2.8)
- `X_pfsdpDeviceFamily` – PFSDP device family (see `device_family` in section 2.3)

A Migrating from R2000 to R2300

This chapter provides additional information on migrating R2000 PFSDP applications to R2300 devices.

A.1 Functional comparison

The following table gives an overview about functional differences between R2000 and R2300 devices, which are relevant for the PFSDP communication:

functionality	R2000 devices	R2300 devices
Field of view	360°	100°
Scan layers	single-layer	multi-layer (4)
Scan rate	up to 50 Hz	50 Hz / 100 Hz
Scan direction	`cw` / `ccw`	`ccw` only
Sampling rate	up to 230 kHz	90 kHz (constant)
Lens contamination monitor (LCM)	yes	no
Switching I/O	yes (up to 4)	no
User interface	display / buttons / status LEDs	status LEDs

Please note:
Client applications can use the commands `get_protocol_info` (section 1.2.8) and `list_parameters` (section 2.2.1) as well as the parameter `device_family` (section 2.3.2) to identify the connected device and available commands. Additionally several device capabilities can be read from the device as well. Please refer to section 2.4 for details.

A.2 PFSDP command implementation

A.2.1 Commands available on R2000 devices only

The following commands are not available on R2300 devices:

- `request_handle_tcp`
- `get_iq_parameter`
- `set_iq_parameter`
- `list_iq_parameters`

A.2.2 Commands available on R2300 devices only

There are currently no commands that are specific to R2300 devices.

A.2.3 Error handling for command requests

When setting parameters with commands like `set_parameter` the arguments are processed one after another, not just after checking all values for validity. This processing is aborted whenever an invalid value or unknown argument name is encountered. Additionally, the command return value `error_text` in most cases does not indicate the argument that was processed when the problem came up. To determine the actual cause of an error response, the arguments have to be tried one at a time.

Please note:
As the R2300 has less computing resources, it does not perform as many checks regarding the validity of requests, e.g. when setting parameters. Some actually questionable settings might be accepted (maybe ignored) without reporting an error, whereas the R2000 would reject them.

A.3 PFSDP parameter implementation

A.3.1 Measurement configuration

The R2300 provides some additional parameters for measurement configuration:

- `layer_enable` (see section 2.6.7)
- `measure_start_angle` / `measure_stop_angle` (see section 2.6.6)

These parameters can usually be left at their default values.

A.3.2 Pilot laser

The R2300 features a *pilot laser* that can be used as alignment aid (see section 2.7). This feature can be configured by a set of new parameters:

- `pilot_laser`
- `pilot_start_angle`
- `pilot_stop_angle`

These parameters can usually be left at their default values.

A.3.3 User tag and user notes

On R2300 devices the parameter `user_tag` accepts strings with a length of up to 32 *bytes* only (R2000 devices accept up to 32 UTF-8 characters). Additionally, the parameter must not contain `NUL` bytes.

The parameter `user_notes` is not supported by the R2300.

A.3.4 HMI parameters

Since R2300 devices possess neither controls nor a graphical user interface all HMI parameters known from the R2000 device family are not available:

- `hmi_button_lock` and `hmi_parameter_lock`
- `hmi_language`
- `hmi_display_mode`
- `hmi_static_logo`
- `hmi_static_text1` and `hmi_static_text2`
- `hmi_application_bitmap`
- `hmi_application_text1` and `hmi_application_text2`

Please note:
The locator indication functionality is available via `locator_indication` (see section 2.8)

A.4 PFSDP scan data implementation

A.4.1 Connection handling

The R2300 scan data output can be controlled with operations including handle management and watchdog feeding just like for the R2000. It accepts those commands and responds properly. However, due to the smaller system design compared to the R2000, only a *single* scan data connection can be managed at a time.

The scan data connection watchdog (which closes a connection if a client application stopped working), is not implemented. Instead the R2300 simply closes any currently active scan data connection when a new connection is requested using `request_handle_udp` (see section 3.3.1).

The following watchdog-related connection parameters are available for compatibility purposes only:

- `watchdog`: accepts the setting `off` only
- `watchdogtimeout`: accepts the value `60000` only

Please note:
The command `feed_watchdog` is available for compatibility purposes, but eventually has no effect.

A.4.2 Scan data output customization

The following scan data connection parameters (see section 3.2.3) are not supported by R2300 devices and are available for compatibility purposes only:

- `skip_scans`: accepts the value `0` only
- `packet_crc`: accepts the setting `off` only

A.4.3 Scan data packet types

The R2300 currently supports the new scan data packet type 'C1' (see section 3.4.4) only. It is largely identical to the R2000 packet type 'C' except for two new fields `layer_index` and `layer_inclination` which have been added to the scan data header behind the `scan_number` and `packet_number` fields.

A.5 Filter-based scan data processing

The R2300 does currently not support the *remission* filter available on R2000 devices. Consequently, the parameter `filter_type` (see section 4.3.2) cannot be set to the value `remission` and the related parameter `filter_remission_threshold` is not available.

B Troubleshooting the Ethernet communication

This chapter contains some basic suggestions for troubleshooting issues concerning the R2300 Ethernet communication.

B.1 Checking the Ethernet setup

In case of communication problems, first ensure a working Ethernet connection between PC and sensor. Please consider the following steps:

- **Sensor IP configuration**
  Check the current IP configuration of the sensor by inspecting SSDP notifications sent by the device during startup (see section 5.1). In case of an improper IP setup try to reset the device into AutoIP mode using the recovery method described in the user manual.
- **Ethernet connection**
  Use the network utility *ping* to verify the network connection between sensor and PC. The sensor will reply to all ping requests it receives. If *ping* does not receive any replies, re-check the IP configuration of your client PC and the sensor. Make sure the IP addresses of both devices are within the same subnet.
- **Electrical connection**
  In case of connectivity problems, check the link status and link speed of the sensor, the client PC and any network infrastructure device (router, switch, etc.) in-between to rule out electric connection issues. For maximum reliability, try to use a direct cable-based Ethernet connection between sensor and PC. The sensor supports Auto-MDIX – a cross-over Ethernet cable is not required.

B.2 Debugging using a web browser

If basic network connectivity has been established, verify that the HTTP command interface is operational with a standard web browser. Please consider these steps:

- **Proxy settings**
  Make sure that no proxy is used when accessing the sensor. In the browser settings, either completely disable any proxy or add a proxy exception for the sensor IP address.
- **HTTP access**
  Try to access the sensor via the following URL:

  ```
  http://<sensor IP address>/cmd/protocol_info
  ```

  This command should return some basic protocol information (see section 1.2.8). If this is not the case, re-check your proxy settings and Ethernet setup (see above).
- **HTTP commands**
  You can test the syntax and effect of any HTTP command used in your application software just by sending the command from a web browser. The web browser will display the response received from the sensor – making it easy to review any potential error messages. Furthermore, after changing sensor settings with the `set_parameter` command (see section 2.2.3), it might be helpful to read back all parameters using the command `get_parameter` (see section 2.2.2).

B.3 Debugging using Wireshark

For complex communication issues it is highly recommended to use the free network traffic analysis tool *Wireshark* [14] to sniff and record the Ethernet communication between the client software and the R2300 sensor.

For example, this can be very helpful for:

- Checking the content of HTTP messages and the corresponding replies
- Checking order and time behavior of HTTP commands
- Checking time behavior of scan data output (TCP or UDP)

In case you contact your sensor support representative about a specific communication issue, it is highly recommended to have a Wireshark log file (`.pcap`) at hand for examination by the technical support organisation.

C Protocol version history

C.1 Protocol version 1.05

First public release.

Note: This protocol version is implemented by R2300 firmware v1.00 or newer.

D Document change history

D.1 Release 2022-08 (protocol version 1.05)

Document update for R2300 firmware v1.00.

Notable extensions:

- Section 2.3.2: Added device family definition for R2300 single-layer devices.
- Section 2.4.1: Added parameters `layer_count` and `layer_inclination`.
- Section 2.4.2: Added feature flags `scan_data_filter` and `scan_data_filter_moving`.
- Section 3.1: Added definition of term *frame rate*.
- Chapter 4: Added documentation on filter-based scan data processing.

Notable changes:

- Section 2.5.1: Updated default values for `ip_address_current` and `subnet_mask_current`
- Section 3.1.5: Removed references to synchronized timestamps (not available on R2300 devices)
- Section 3.4.4: Fixed incorrectly documented value of field `packet_type`.
- Appendix A.4.1: Added note on compatibility with command `feed_watchdog`.
- Replace term *scan frequency* with term *scan rate* in whole document.
- Various minor textual and cosmetic updates

D.2 Release 2020-10 (protocol version 1.05)

First release.

Index for commands and parameters

This index provides a quick reference for all commands and parameters defined by this communication protocol.

Generic commands (URI)
- factory_reset, 15
- feed_watchdog, 50
- get_parameter, 14
- get_protocol_info, 8, 9, 48
- get_scanoutput_config, 34
- list_parameters, 13, 48
- reboot_device, 15, 19
- release_handle, 32
- request_handle_udp, 29, 31, 50
- reset_parameter, 14, 15
- set_parameter, 14, 48
- set_scanoutput_config, 29, 31, 33, 34
- start_scanoutput, 32, 34
- stop_scanoutput, 33, 34

Global parameters (sensor)
- angular_fov, 18, 22
- angular_resolution, 18
- device_family, 17, 48
- emitter_type, 18
- feature_flags, 18
- filter_error_handling, 43–45
- filter_maximum_margin, 43–45
- filter_remission_threshold, 50
- filter_type, 42, 44, 50
- filter_width, 42–45
- gateway, 19
- gateway_current, 19
- hmi_application_text2, 49
- hmi_parameter_lock, 49
- hmi_static_text2, 49
- ip_address, 19
- ip_address_current, 19
- ip_mode, 19
- ip_mode_current, 19
- layer_count, 18, 26
- layer_enable, 20, 22, 23, 49
- layer_inclination, 18, 26
- locator_indication, 23, 49
- mac_address, 19
- max_connections, 18, 31
- measure_start_angle, 20–23, 29, 42, 49
- measure_stop_angle, 20–23, 29, 42, 49
- operating_mode, 20
- operation_time, 24
- operation_time_scaled, 24
- part, 17, 47
- pilot_laser, 22, 23, 49
- pilot_start_angle, 22, 23, 49
- pilot_stop_angle, 22, 23, 49
- power_cycles, 24
- product, 17, 47
- radial_range_max, 18, 27
- radial_range_min, 18, 27
- radial_resolution, 18, 27
- revision_fw, 17
- revision_hw, 17
- samples_per_scan, 20–22, 26, 30, 37, 39
- sampling_rate_max, 18
- sampling_rate_min, 18
- scan_direction, 20, 21
- scan_frequency, 20, 21, 26, 37, 39
- scan_frequency_max, 18
- scan_frequency_measured, 20
- scan_frequency_min, 18
- serial, 17, 47
- status_flags, 24, 25
- subnet_mask, 19
- subnet_mask_current, 19
- system_time_raw, 24, 28
- temperature_current, 24
- temperature_max, 24
- temperature_min, 24
- up_time, 24
- user_tag, 17, 49
- vendor, 17, 47

Scan data output parameters
- address, 31
- handle, 6, 31–35
- max_num_points_scan, 22, 30, 31, 34, 37, 39, 42
- packet_type, 31, 34
- port, 31
- start_angle, 22, 29–31, 34, 42

Protocol version 1.05 Document release 2022-08

PEPPERL+FUCHS

References

[1] RFC-791: Internet Protocol Specification
http://tools.ietf.org/html/rfc791

[2] RFC-1305: Network Time Protocol (Version 3)
http://tools.ietf.org/html/rfc1305

[3] RFC-2046: Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types
http://tools.ietf.org/html/rfc2046

[4] RFC-2069: An Extension to HTTP : Digest Access Authentication
http://tools.ietf.org/html/rfc2069

[5] RFC-2616: Hypertext Transfer Protocol – HTTP/1.1
http://tools.ietf.org/html/rfc2616

[6] RFC-2617: HTTP Authentication: Basic and Digest Access Authentication
http://tools.ietf.org/html/rfc2617

[7] RFC-3629: UTF-8, a transformation format of ISO 10646
http://tools.ietf.org/html/rfc3629

[8] RFC-3986: Uniform Resource Identifier (URI)
http://tools.ietf.org/html/rfc3986

[9] RFC-4648: The Base16, Base32, and Base64 Data Encodings
http://tools.ietf.org/html/rfc4648

[10] RFC-7159: The JavaScript Object Notation (JSON) Data Interchange Format
http://tools.ietf.org/html/rfc7159

[11] Online CRC32 calculation of an example buffer
http://crccalc.com/?crc=0x010x020x030x040x050x060x070x08&method=crc32&datatype=hex

[12] Libwww: the W3C Protocol Library
http://www.w3.org/Library/

[13] libcurl: free and easy-to-use client-side URL transfer library
http://curl.haxx.se/libcurl/

[14] Wireshark: free network protocol analyzer for Unix and Windows
http://www.wireshark.org/

[15] Simple Service Discovery Protocol (Draft v1.03)
https://tools.ietf.org/html/draft-cai-ssdp-v1-03

[16] Zero Configuration Networking (Zeroconf)
http://www.zeroconf.org/

FACTORY AUTOMATION – SENSING YOUR NEEDS

Worldwide Headquarters
Pepperl+Fuchs GmbH · Mannheim · Germany
E-mail: fa-info@pepperl-fuchs.com

USA Headquarters
Pepperl+Fuchs Inc. · Twinsburg, OH · USA
E-mail: fa-info@us.pepperl-fuchs.com

Asia Pacific Headquarters
Pepperl+Fuchs Pte Ltd · Singapore
Company Registration No. 199003130E
E-mail: fa-info@sg.pepperl-fuchs.com

#66013 clean – 2022-08-15 14:50

www.pepperl-fuchs.com

Subject to modifications without notice
Copyright Pepperl+Fuchs · Printed in Germany