

Data format description

Compact format and MSGPACK format for transmitting measurement data

SICK
Sensor Intelligence.

```
void WindowWorker::T1000(const boost::posix_time::time_duration cycleTime = boost::posix_time::time_duration(0, 0, 0, 1000000))  
// Initialize Window  
boost::posix_time::time_duration cycleTime = boost::posix_time::time_duration(0, 0, 0, 1000000)  
if (!m_lastTime.is_special()) {  
    cycleTime = time - m_lastTime; } m_lastTime = time;  
double cycleSec = double(cycleTime.total_microseconds()) / 1000000.  
// Speed Information  
if (m_speedAvailable && m_useSpeedForDistance) {  
    double deltaX = m_currentSpeed * cycleSec; m_windowDistance += deltaX;  
}  
// Window Position  
double mowedWindow = m_windowDistance - m_lastCompensatedWindow;  
double deltaAng = (m_WindowAvailable ? mowedWindow : 0);  
Point2D movement(Point2D::fromPolar(movedDist, deltaAng));
```

Contents

1	Glossary.....	3
2	General information on the transmission of measurement data.....	4
3	Parameterization of the data output via telegrams.....	5
3.1	Selecting the data output format.....	5
3.2	Communication settings for data transmission.....	5
3.3	Activating the data transmission.....	5
3.4	Example.....	5
4	MSGPACK format.....	7
4.1	Framing.....	7
4.2	MSGPACK keywords.....	7
4.3	Serialization of a segment.....	9
4.3.1	Notation used.....	9
4.3.2	Serialization of the “Scan segment” class.....	10
4.3.3	Serialization of the “Scan” class.....	11
4.3.4	Serialization of arrays.....	13
5	Compact format.....	14
5.1	Framing.....	14
5.2	Division of a segment into modules.....	15
5.3	Serialization of modules.....	15
5.3.1	Meta data.....	16
5.3.2	Measurement data.....	19
5.4	IMU format.....	21
6	Definitions applicable to both data formats.....	23
6.1	Azimuth angle.....	23
6.2	Segment counter.....	23
6.3	Representation of RSSI values.....	24
6.4	Representation of bit fields.....	24
6.5	Representation of beam characteristics (“Properties”).....	25
7	Behavior of serialization for data reduction.....	26
7.1	Behavior in relation to the number of available echoes.....	26
7.2	Behavior when restricting the azimuth angular range.....	26
7.3	Behavior when reducing the available layers.....	27

1 Glossary

Designation	Explanation
RSSI	<p>RSSI (received signal strength indicator) is defined as an indicator of the strength of the received signal.</p> <p>Context:</p> <ul style="list-style-type: none"> • Small RSSI value = low signal strength • Large RSSI value = high signal strength <p>The magnitude of the RSSI values is not standardized and can vary from device to device.</p>
Modules	A module is an object that creates/provides data for different layers
layer	A traditional 2D LiDAR has a layer at 0°. Multi-layer scanners can have multiple layers (multiScan => 16 layers).
Beam	Is a measurement beam with specific properties such as distance, remission, ...
Scan	Collection of beams in the azimuth direction of a layer.
Scan segment	A scan segment is defined as a collection of scans that represent a portion of a frame. All scans in a segment have different elevation angles (i.e., they belong to different layers in the case of a multi-layer LiDAR sensor). The angular range in azimuth may (but need not) be different for each scan in a segment. For multi-layer LiDAR sensors, a scan segment is typically used to combine scans that were acquired at the same time or that have the same azimuth range.
Frame	<p>A frame is defined as the data acquired within the entire field of view of a LiDAR sensor (e.g., 360° for rotating LiDAR sensors such as the multiScan)</p> <p>multiScan example:</p> <ul style="list-style-type: none"> • 1 frame consists of 12 segments • 1 segment comprises n layers = n scans (multi-Scan136 example: 16 layers = 16 scans) • 1 scan comprises several beams
Azimuth angle	Theta
Elevation angle	Phi

2 General information on the transmission of measurement data

The measurement data are transmitted segment by segment, i.e., each transmitted data package (for example a UDP packet or TCP packet) contains a segment (cf. the Segmented data output section in the operating instructions). Each segment can be interpreted separately, i.e., it is not necessary to collect all segments in a frame (=all measurement data recorded in one revolution) to start processing. This makes it possible to reduce the latency between the generation of the measurement data and the processing of that data on the client side.

Two formats are available for the transmission of measurement data, which will be referred to in the following as MSGPACK format and Compact format.

The MSGPACK format encodes the measurement data according to the MSGPACK standard (see also www.msgpack.org), which has the advantage that the data packages can be easily parsed using the standard libraries available for numerous programming languages. The MSGPACK format is self-describing. Each data field is described by a keyword, so it is easy to determine which data field is currently being read without having to know the exact structure of the data.

In the Compact format, the measurement data of the sensor are represented as compactly as possible. The individual fields are no longer self-describing as with MSGPACK, but only a string of bytes is transmitted. The structure of the transmitted data package must be known to the user in advance. This has the advantage that as little bandwidth as possible is used on the data line and that a very efficient interpretation of the data is possible by copying the transmitted byte sequence into a structure using a single command (e.g., memcpy in the programming language C/C++).

3 Parameterization of the data output via telegrams

Various telegrams for parameterization and activation of the data output are described below. An example is provided in [section 3.4](#) for the communication protocol CoLa A only, telegrams via other communication interfaces can be found in the Telegram listing under www.sick.com/8014631.

3.1 Selecting the data output format

Variable name: ScanDataFormat

Parameters:

Table 1: Selecting the data output format

Name	Type	Values: Meaning
-	Enum	1: MSGPACK , 2: Compact

3.2 Communication settings for data transmission

Variable name: ScanDataEthSettings

Parameters:

Table 2: Communication settings for data transmission

Name	Type	Sensor	Values: Meaning
Protocol	Enum	multiScan1xx	1: UDP
		-	2: TCP (reserved)
IPAddress	Array of unsigned short int	multiScan1xx	IP address of the data recipient, each array element stands for one digit of the IP address, e.g. {192,168,0,102} stands for the IP address 192.168.0.102
Port	Unsigned int	multiScan1xx	Port of the recipient to which the data is sent

3.3 Activating the data transmission

Variable name: ScanDataEnable

Parameter:

Table 3: Activating the data transmission

Name	Type	Values: Meaning
-	Bool	True for active data output, false for deactivated data output

3.4 Example

Configuration and activation of the MSGPACK data output to the client IP address 192.168.0.102 and client port 2115. The commands are specified in the CoLa A dialect and are sent via TCP/IP to the IP address of the sensor on port 2111.

Table 4: Example parameterization of data output

Command (in CoLa A dialect)	Description
<STX>sMN SetAccessMode 3 F4724744<ETX>	Log on to the sensor with the “Authorized Customer” user level.
<STX>sWN ScanDataFormat 1<ETX>	Select MSGPACK as data format.
<STX>sWN ScanDataEthSettings 1 C0 A8 0 66 843<ETX> Alternatively with arguments in decimal notation: <STX>sWN ScanDataEthSettings 1 +192 +168 +0 +102 +2115<ETX>	Data is sent via UDP to the IP address 192.168.0.102 and port 2115. NOTE In CoLa A, integers are usually specified in hexadecimal notation. To use decimal numbers, a sign (+ or -) must be placed in front of the number.
<STX>sWN ScanDataEnable 1<ETX>	Activate data output.
<STX> sWN ScanDataEnable 0 >ETX>	Deactivate data output.
<STX>sMN Run 1<ETX>	Log out of the sensor. NOTE The parameterization only becomes active after logging off from the sensor.











	30	<STX>sMN SetAccessMode 3 F4724744<ETX>
	21	<STX>sAN SetAccessMode 1<ETX>
	22	<STX>sWN ScanDataFormat 1<ETX>
	20	<STX>sWA ScanDataFormat<ETX>
	42	<STX>sWN ScanDataEthSettings 1 C0 A8 0 66 843<ETX>
	25	<STX>sWA ScanDataEthSettings<ETX>
	22	<STX>sWN ScanDataEnable 1<ETX>
	20	<STX>sWA ScanDataEnable<ETX>
	11	<STX>sMN Run 1<ETX>
	11	<STX>sAN Run 1<ETX>

Figure 1: Example parameterization of data output

4 MSGPACK format

4.1 Framing

The data packages transmitted in MSGPACK format are enclosed within a frame (see [figure 2, page 7](#)):

The actual MSGPACK payload data is preceded by 4 <STX> characters (hex code 0x02) and the size of the actual payload data (without the checksum at the end) in bytes as a uint32 value. Following the MSGPACK data is a CRC32 checksum calculated on the MSGPACK data only (without the <STX> characters and packet size). The little-endian representation is used for both the size of the payload and the checksum.

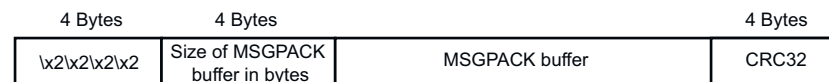


Figure 2: Framing in the MSGPACK format

4.2 MSGPACK keywords

To reduce the bandwidth on the transmission line, the keywords used in MSGPACK are encoded as uint8 values. The bandwidth savings from this are significant, but the user must interpret the read uint8 keywords according to the following table. The description of the keywords in the table is for overview purposes only. A detailed description can be found in the sections referred to in the respective table rows. Exactly the same names for the keywords are used there.



NOTE

The data packages can be read using standard MSGPACK parsers after removing the framing. For some parsers, options must be set to allow uint8 values as keywords. For the Python msgpack module, for example, the option `strict_map_key=False` must be set:

```
unpacked = msgpack.unpackb(msgpackValue, strict_map_key=False)
```

Table 5: Used MSGPACK keywords and associated uint8 codes

Keyword name	Uint8 value	Description
classname	0x10	Keyword for the Scan (see "Serialization of the "Scan" class", page 11) and ScanSegment (see "Serialization of the "Scan segment" class", page 10) classes represented in the data
data	0x11	Keyword for the data part, which belongs to the Array, Scan or ScanSegment classes, see "Serialization of the "Scan" class", page 11 ; see "Serialization of the "Scan segment" class", page 10 ; see "Serialization of arrays", page 13 .
numOfElems	0x12	Number of elements in an array, see "Serialization of arrays", page 13
elemSz	0x13	Size of an array element in bytes, see "Serialization of arrays", page 13
endian	0x14	Keyword describing the endianness of the array elements, see "Serialization of arrays", page 13

Keyword name	Uint8 value	Description
elemTypes	0x15	Keyword for the type of array elements, see "Serialization of arrays", page 13
Little	0x30	Keyword for endianness "little".
float32	0x31	Data type float32
uint32	0x32	Data type uint32
uint8	0x33	Data type uint8
uint16	0x34	Data type unit16
ChannelTheta	0x50	Data channel with azimuth angles, see "Serialization of the "Scan" class", page 11
ChannelPhi	0x51	Data channel with elevation angles, see "Serialization of the "Scan" class", page 11
DistValues	0x52	Channel with distance values, see "Serialization of the "Scan" class", page 11
RssiValues	0x53	Channel with RSSI values, see "Serialization of the "Scan" class", page 11
PropertiesValues	0x54	Channel with further properties of a measurement beam, see "Serialization of the "Scan" class", page 11 and see "Representation of beam characteristics ("Properties")", page 25
Scan	0x70	Keyword for the "Scan" class, see "Serialization of the "Scan" class", page 11
TimestampStart	0x71	Start time stamp of a scan, see "Serialization of the "Scan" class", page 11
TimestampStop	0x72	Stop time stamp of a scan, see "Serialization of the "Scan" class", page 11
ThetaStart	0x73	Azimuth start angle of a scan, see "Serialization of the "Scan" class", page 11
ThetaStop	0x74	Azimuth stop angle of a scan, see "Serialization of the "Scan" class", page 11
ScanNumber	0x75	Number of a scan, see "Serialization of the "Scan" class", page 11
ModuleId	0x76	ModuleID of a scan, see "Serialization of the "Scan" class", page 11
BeamCount	0x77	Number of beams in a scan, see "Serialization of the "Scan" class", page 11
EchoCount	0x78	Number of echoes in a scan, see "Serialization of the "Scan" class", page 11
ScanSegment	0x90	Keyword for the "ScanSegment" class, see "Serialization of the "Scan segment" class", page 10
SegmentCounter	0x91	Segment number of a segment, see "Serialization of the "Scan segment" class", page 10
FrameNumber	0x92	Frame number of a segment, see "Serialization of the "Scan segment" class", page 10
Availability	0x93	Availability of a segment, see "Serialization of the "Scan segment" class", page 10
SenderId	0x94	SenderID of a segment, see "Serialization of the "Scan segment" class", page 10
SegmentData	0x96	Array with the actual measurement data for each layer, see "Serialization of the "Scan segment" class", page 10

Keyword name	Uint8 value	Description
LayerId	0xA0	Array with layer IDs, see "Serialization of the "Scan segment" class", page 10
TelegramCounter	0xB0	Telegram counter, see "Serialization of the "Scan segment" class", page 10

4.3 Serialization of a segment

Each data package transmits one segment enclosed within a frame as per section [section 4.1](#). A segment contains various fields, which are described in [see table 6](#). The actual measurement data is located in the **SegmentData** field, which contains one element of the **Scan** class ([see "Serialization of the "Scan" class", page 11](#)) for each layer of the sensor ([see figure 3, page 9](#)).

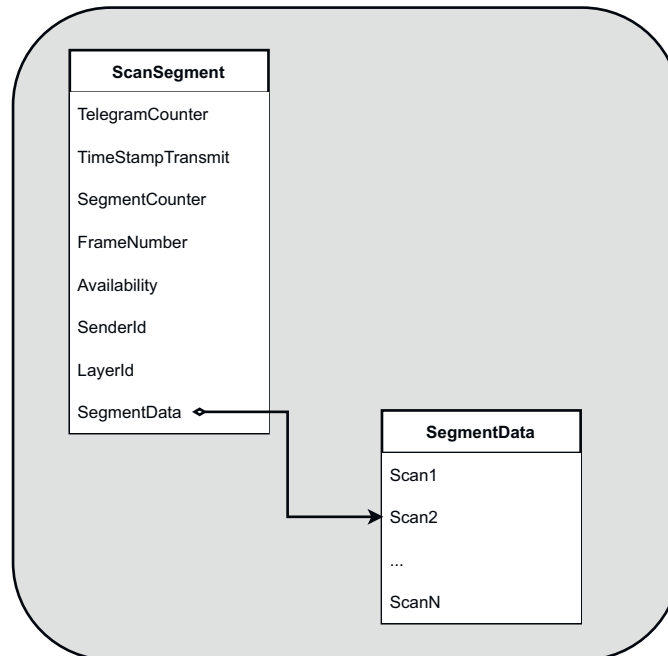


Figure 3: General structure of a segment. In addition to some metadata, the `ScanSegment` class contains an array named `SegmentData` that contains an object of type `Scan` for each layer of the sensor.

The data of a segment are encoded in MSGPACK format. The following exception should be noted: Measurement data in arrays such as distance, RSSI, angle etc. are binary coded here to allow easier serialization ([see "Serialization of arrays", page 13](#)).

4.3.1 Notation used

The following notes on the notation used relate to the description of the structures encoded in MSGPACK:

- `Msgpack_map_header` is used to denote the header of a msgpack map as per this specification: <https://github.com/msgpack/msgpack/blob/master/spec.md#map-format-family>
- The keyword names and other names used in the structures correspond to those from [see table 5, page 7](#).
- Angle brackets are used to specify placeholders for values referred to in the respective declarations, e.g., `<numOfElems>` for the number of elements of an array.

- Keywords printed in bold refer to substructures of a type, e.g., other types or arrays.
- Although a structure similar to JSON is used for the description in this document, the data is however encoded according to the MSGPACK specification: <https://github.com/msgpack/msgpack/blob/master/spec.md>

4.3.2 Serialization of the “Scan segment” class

The ScanSegment class is represented as a nesting of MSGPACK maps as follows:

```
msgpack_map_header {
  "classname": ScanSegment,
  "data":
  msgpack_map_header {
    "TelegramCounter": <telegramCounter>,
    "TimeStampTransmit": <timeStampTransmit>,
    "SegmentCounter": <segmentCounter>,
    "FrameNumber": <frameNumber>,
    "Availability": <availability>,
    "SenderId": <senderId>,
    "LayerId": layerIdVector,,
    "SegmentData", segmentData
  }
}
```

Definition: Definition of the ScanSegment class

The meaning of the individual fields is shown in the following table.

Table 6: Description of the attributes of the ScanSegment class

Name	Type	Description
TelegramCounter	MSGPACK int ¹⁾	Counts all telegrams with measurement data sent in MSGPACK format since switching on the device. The counter starts at 1.
TimeStampTransmit	MSGPACK int ¹⁾	Sensor system time in μ s since 1.1.1970 00:00 in UTC.
SegmentCounter	MSGPACK int ¹⁾	Segment counter as described in section.
FrameNumber	MSGPACK int ¹⁾	Counts the number of full revolutions since the device was started.
Reserved		
SenderId	MSGPACK int ¹⁾	Device serial code. It can be used to detect on the recipient which sensor the data was sent from.
LayerId	MSGPACK array ²⁾ of int ¹⁾	Array of layer indices. The layer indices start at 1 and increase with decreasing elevation angle .
SegmentData	MSGPACK array ²⁾ of scans	Array of elements of the Scan class (see "Serialization of the “Scan” class", page 11) that contain the actual measurement data. The following applies: The scan at position i has the layer number LayerId[i].

¹⁾ github.com/msgpack/msgpack/blob/master/spec.md#int-format-family

²⁾ github.com/msgpack/msgpack/blob/master/spec.md#array-format-family

NOTE | The arrays LayerId and SegmentData are MSGPACK arrays as per the MSGPACK specification and are not represented like arrays containing measurement data (see "Serialization of arrays", page 13).

4.3.3 Serialization of the “Scan” class

The Scan class is represented as a nesting of MSGPACK maps as follows:

```
msgpack_map_header {
  "classname": Scan,
  "data":
  msgpack_map_header {
    "TimeStampStart": <timeStampStart>,
    "TimeStampStop": <timeStampStop>,
    "ThetaStart": <thetaStart>,
    "ThetaStop": <thetaStop>,
    "ScanNumber": <scanNumber>,
    "ModuleId": <moduleId>,
    "ChannelTheta": <channelTheta>,
    "ChannelPhi": <channelPhi>,
    "DistValues": <distValues>,
    "RssiValues": <rssiValues>,
    "PropertyValues": <propertyValues>,
    "BeamCount": <beamCount>,
    "EchoCount": <echoCount> }
}
```

Definition: Definition of the “Scan” class. The fields highlighted in gray are optional, i.e., they are not necessarily present in the structure.

The meaning of the individual fields is shown in the following table:

Table 7: Description of the attributes of the Scan class

Name	Type	Description
TimeStampStart	MSGPACK int ¹⁾	Acquisition time of the first beam of the scan in μ s. The device's internal time base is used or, if the sensor offers the feature, the time set externally.
TimeStampStop	MSGPACK int ¹⁾	Acquisition time of the last beam of the scan in μ s. The device's internal time base is used or, if the sensor offers the feature, the time set externally.
ThetaStart	MSGPACK int ¹⁾	Azimuth angle of the first beam of the scan in radians.
ThetaStop	MSGPACK int ¹⁾	Azimuth angle of the last beam of the scan in radians.
ScanNumber	MSGPACK int ¹⁾	Not used.
ModuleId	MSGPACK int ¹⁾	Number of the physical module that generated the data. In the case of the multiScan, for example, this is one of the two measuring modules.

Name	Type	Description
ChannelTheta	array of float32	Array of azimuth angles of a specific beam in radians. See also section 6.1 . The encoding of the data in this array is described in section 4.3.4 . The user can configure whether the array is present in the data structure.
ChannelPhi	array of float32	Array with the elevation angle (cf. operating instructions, Coordinate system section) of the scan in radians. For single layer and multi-layer sensors, this array contains only one element. The encoding of the data in this array is described in section 4.3.4 . The user can configure whether the array is present in the data structure.
DistValues	array of (array of float32)	Array containing an array of distance values for each echo of the scan. The number of sub-arrays corresponds to the number of echoes in the scan, see also the “EchoCount” field below. The distance values are specified in mm. The encoding of the data in this array is described in section 4.3.4 . The user can configure whether the array is present in the data structure. The structure of the nested array is illustrated in figure 4 .
RssiValues	array of (array of uint16)	Array containing an array of RSSI values for each echo of the scan. The number of sub-arrays corresponds to the number of echoes in the scan, see also the “EchoCount” field below. For the representation of RSSI values, see " Representation of RSSI values ", page 24. The encoding of the data in this array is described in section 4.3.4 . The user can configure whether the array is present in the data structure. The structure of the nested array is illustrated in figure 4 .
PropertyValues	array of uint8	Array with additional properties, for example “reflector”, for each beam of a scan. See section 6.4 for details. The encoding of the data in this array is described in section 4.3.4 . This array is optional.
BeamCount	MSGPACK int ¹⁾	Number of beams in the current scan.
EchoCount	MSGPACK int ¹⁾	Number of echoes in the current scan.

1) github.com/msgpack/msgpack/blob/master/spec.md#int-format-family

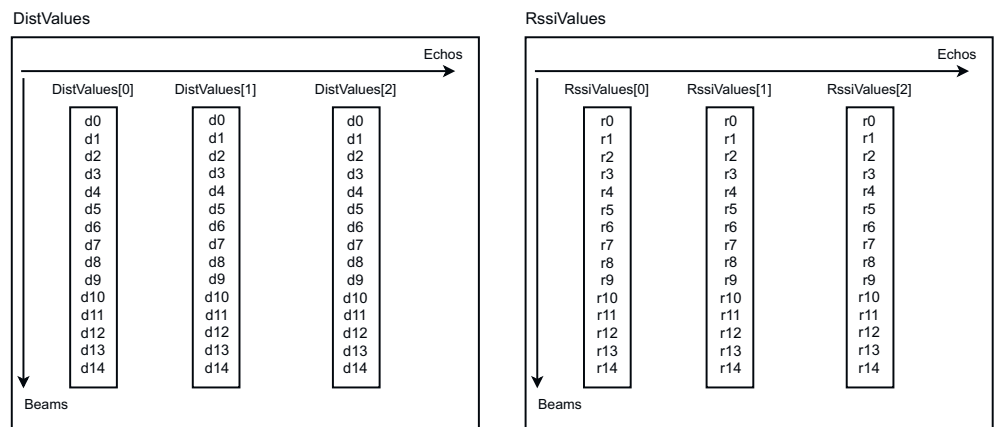


Figure 4: Example of the structure of the `DistValues` and `RssiValues` arrays. Shown here are data for a scan with 3 echoes and 15 beams.

4.3.4 Serialization of arrays

Arrays of measurement data (ChannelTheta, ChannelPhi, DistValues, RssiValues and PropertyValues from [see table 7, page 11](#)) are encoded as follows:

```
msgpack_map_header {
  "numOfElems": <number of array elements>,
  "elemSz", < size of one array element in bytes>,
  "endian", little,
  "elemTypes", typesArray, ,
  "data": binaryData
}
```

Definition: Definition of the “Array” class

The meaning of the individual fields of the Array class are shown in the following table.

Table 8: Description of the attributes of the “Array” class

Name	Description
numOfElems	Number of array elements
elemSz	Size of an array element in bytes, e.g., 4 for float32 or 1 for uint8.
endian	The value is always “little”
elemTypes	Array of element types. Only arrays with one element are supported here. Example: {float32} for an array of float32 values. To represent an array of tuples, elemTypes would therefore have multiple elements. This is not relevant, however.
data	Actual payload. The data is encoded as a byte array in the MSGPACK int ¹⁾ and can be copied directly into a structure/array after parsing the MSGPACK structure, taking into account the endianness and the data type.

1) github.com/msgpack/msgpack/blob/master/spec.md#int-format-family

5 Compact format

5.1 Framing

The data packages transmitted in Compact format are enclosed in a frame consisting of a header before the actual payload and a checksum after the payload.

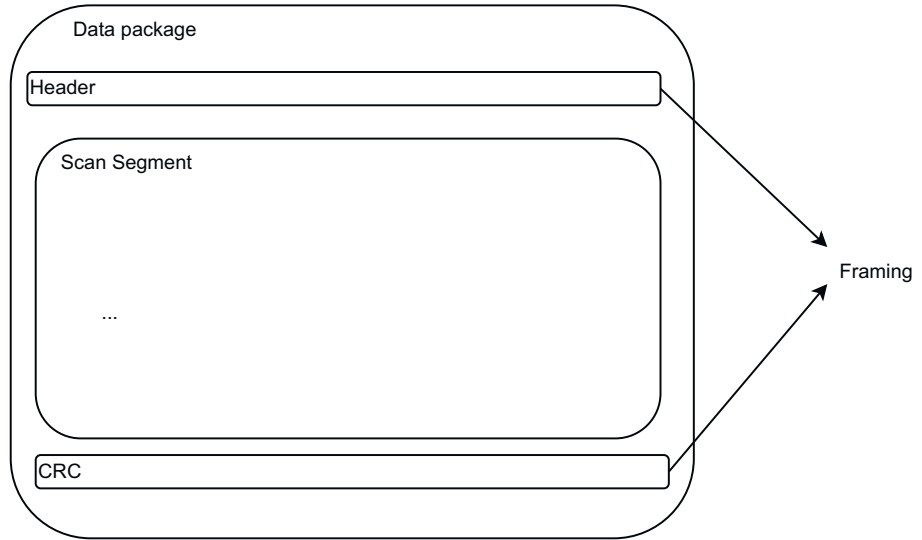


Figure 5: Framing in the Compact format, consisting of a header before the payload and a checksum after the payload.

The structure of the frame header is shown in the following figure. The following table explains the meaning of the individual fields of the header.

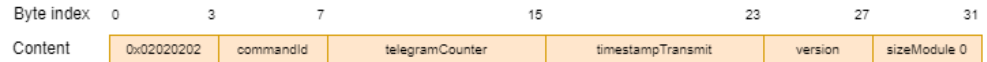


Figure 6: Structure of the Compact frame header.

Table 9: Description of the fields of the Compact frame header.

Name	Size	Type	Description
startOfFrame	4 bytes	uint32	Four <STX> characters (hex code 0x02): \x2\x2\x2\x2
commandId	4 bytes	uint32	Type of the transmitted telegram. To transmit primary data, the commandId is 1.
telegramCounter	8 bytes	uint64	Counts all telegrams sent since the device was switched on. The counter starts at 1.
timeStampTransmit	8 bytes	uint64	Sensor system time in μ s since 1.1.1970 00:00 in UTC. If a time server is being used, the relevant set time will be used.
telegramVersion	4 bytes	uint32	Version of the telegram with the commandId used. For the telegram for serialization of primary data (commandId 1), only telegramVersion 3 is currently used.
sizeModule0	4 bytes	uint32	Size of the first module to be read. See section 5.2 for the definition of modules and for notes on how to extract them from the data packages.

The header always has a fixed size of 32 bytes.

The CRC32 checksum that follows the payload is (in contrast to the MSGPACK format, see section [section 4.1](#)) is calculated over the entire data package, i.e., over the header and the serialized scan segment.

All values in the header and the checksum are encoded as little endian.

5.2 Division of a segment into modules

An important design feature of the Compact format is that the data of the different layers are grouped into modules. Both data generated by different physical measuring modules (e.g., the different measuring modules in a multiScan1xx) and data that differ in scope or angular resolution (e.g., HighResolution layers vs. layers with 1° angular resolution in the case of the multiScan136) are assigned to different modules. The following figure shows an example of this for the multiScan136.

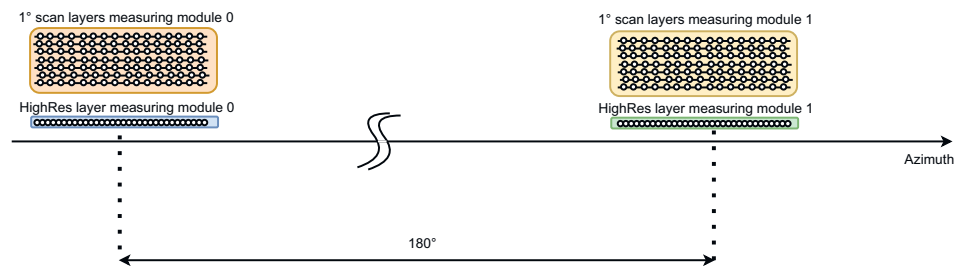


Figure 7: For the multiScan136, a segment consists of four modules, which are shown as colored boxes in the figure: For each of the two physical measurement modules 0 and 1 there are two modules: One for the layers with 1° angular resolution, and one for the HighRes layer with $1/8^\circ$ angular resolution.

The modules available in a data package can vary depending on the configuration of the sensor. It is therefore recommended to proceed module by module when reading the data, as illustrated in the following pseudo code.

Table 10: Pseudocode: Example of reading the individual modules from a data package.

```

Read the 32 Byte Header
Set currentModuleSize = sizeModule0 from the header
As long as currentModuleSize != 0
Read next module of size currentModuleSize
Set currentModuleSize = nextModuleSize from the module just read
(see "Meta data", page 16 )

```

5.3 Serialization of modules

Each module contains the actual measurement data and metadata describing the measurement data:

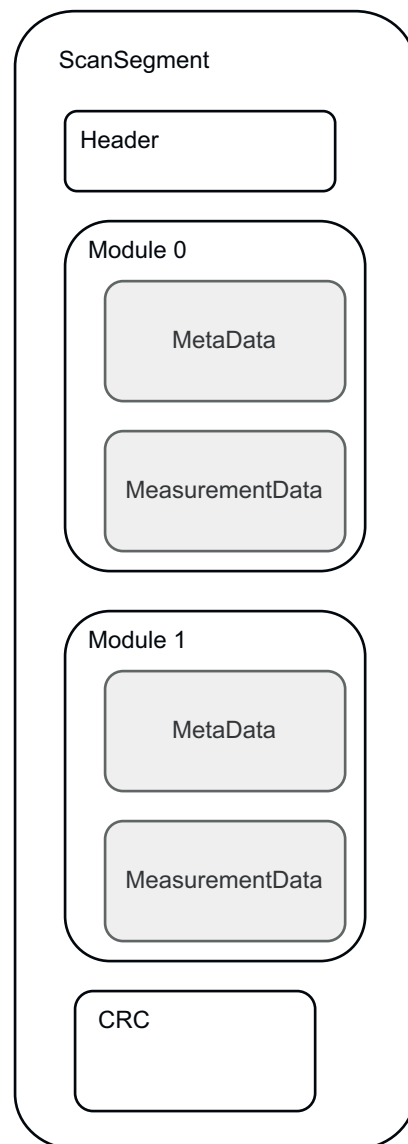


Figure 8: Metadata and measurement data in the individual modules. In the example there are two modules, and in each module the metadata comes first and then the measurement data.

5.3.1 Meta data

This section describes the metadata of a module. The names of the individual data fields in the following table are for orientation purposes only. Since the data are present as a byte sequence only, the names are not used explicitly.

Table 11: Metadata of a module for the Compact format

Name	Size	Type	Description
SegmentCounter	8 bytes	uint64	Segment counter as described in section 6.1 .
FrameNumber	8 bytes	uint64	Counts the number of full revolutions since the device was started.
SenderId	4 bytes	uint32	Device serial code. It can be used to detect on the receiver which sensor the data was sent from.
numberOfLinesInModule	4 bytes	uint32	Number of layers contained in one module, see figure 11 .

Name	Size	Type	Description
NumberOfBeamsPer-Scan	4 bytes	uint32	Number of beams per scan from one layer, see figure 11 . Scans from all layers in a module have the same number of beams, see "Division of a segment into modules", page 15 .
NumberOfEchosPer-Beam	4 bytes	uint32	Number of echoes per beam, see figure 10 .
TimeStampStart	Number of elements * 8 bytes	array of uint64	Array of acquisition times for the first beam of each scan in the current module in μ s. The device's internal time base is used or, if the sensor offers the feature, the time set externally. The length of the array is numberOfLinesInModule.
TimeStampStop	Number of elements * 8 bytes	array of uint64	Array of acquisition times for the last beam of each scan in the current module in μ s. The device's internal time base is used or, if the sensor offers the feature, the time set externally. The length of the array is numberOfLinesInModule.
Phi	Number of elements * 4 bytes	array of float32	Array of elevation angles (cf. operating instructions, Coordinate system section) in radians of each layer in the current module. The length of the array is numberOfLinesInModule.
ThetaStart	Number of elements * 4 bytes	array of float32	Array of azimuth angles in radians for the first beam of each scan of a layer in the current module. The length of the array is numberOfLinesInModule.
ThetaStop	Number of elements * 4 bytes	array of float32	Array of azimuth angles in radians for the last beam of each scan of a layer in the current module. The length of the array is numberOfLinesInModule.
DistanceScalingFactor	4 bytes	float32	This factor is used to scale the distance values in the beam data to allow the display of values over 65,535 mm with 16 bits or alternatively a sub-millimeter resolution. Formula: d_mm_external = DistanceScalingFactor * d d is the distance value contained in the beam data d_mm_external is the distance value in mm which can be derived from a consumer of streaming data from d. If the scaling factor ≥ 1 , integers (1, 2, 3, ...) are always entered so that integer values are used for the conversion.
NextModuleSize	4 bytes	uint32	Size of the next module, or 0 if the current module is the last one. This value is important when reading the data using the principle in table 10 , see figure 9 .
Reserved	1 byte	uint8	-
DataContentEchos	1 byte	uint8	The individual bits of this byte describe which data are available in that part of the measurement data that is acquired per echo, e.g., distance or RSSI, see table 12, page 18 .

Name	Size	Type	Description
DataContentBeams	1 byte	uint8	The individual bits of this byte describe which data are available in that part of the measurement data that is only acquired once per beam, e.g., azimuth angle or beam properties, see table 13, page 19 .
Reserved	1 byte	uint8	Fill byte to ensure the metadata has a 32-bit alignment.

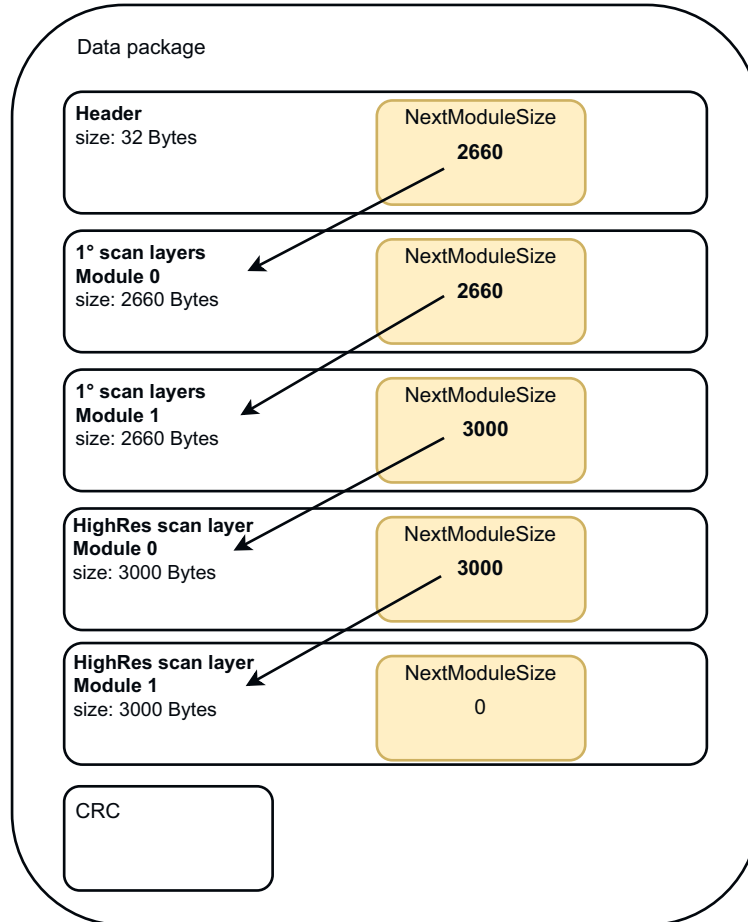


Figure 9: Example of the NextModuleSize field. The modules with the 1° layers are 2,660 bytes in size, the modules with the HighRes layers are 3,000 bytes in size. The value for NextModuleSize in the header as well as in the first serialized module is therefore 2660, and in the two subsequent modules 3000. In the last serialized module the value is 0, because no further module follows.

The bit indices in the DataContentEchos and DataContentBeams bytes are derived as described in [section 6.4](#).

Table 12: Description of the bits of DataContentEchos

Bit index	Value
0	1 if distance data is available, otherwise 0.
1	1 if RSSI data is available, otherwise 0.
2	Reserved
3	Reserved
4	Reserved

Bit index	Value
5	Reserved
6	Reserved
7	Reserved

Table 13: Description of the bits of DataContentBeams

Bit index	Value
0	1 if additional beam properties are available, otherwise 0.
1	1 if azimuth angles per beam are available, otherwise 0.
2	Reserved
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Reserved

5.3.2 Measurement data

Each beam of a scan is represented as a tuple whose elements are represented by the bytes DataContentEchos (table 12) and DataContentBeams (table 13) in the metadata. First are the contents per echo included as described by DataContentEchos. These appear sequentially per echo as shown. This is then followed by the contents included as described by DataContentBeams. The representation of the respective contents is shown in the following table.

Table 14: Representation of the individual measurement data fields

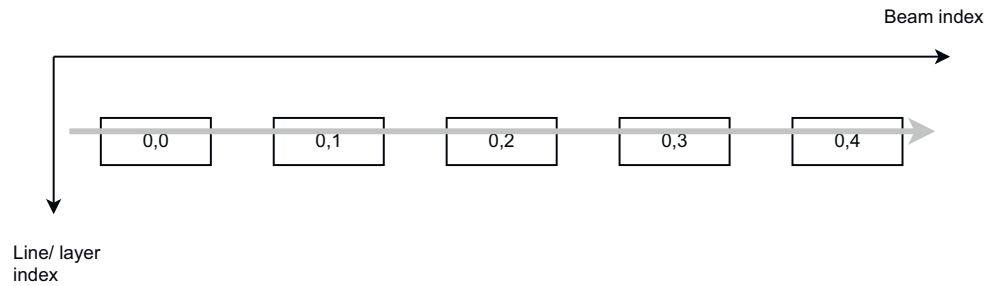
Content	Size	Type	Representation
Distance	2 bytes	uint16	uint16 integers in mm
RSSI	2 bytes	uint16	see "Representation of RSSI values", page 24
Beam characteristics ("Properties")	1 byte	uint8	see "Representation of beam characteristics ("Properties")", page 25
Azimuth angle (theta)	2 bytes	uint16	uint16 Integers, where the following conversion applies: <ul style="list-style-type: none"> • a_uint: Angle value as integer • a_rad: Angle value in radians. • $a_{rad} = (a_{uint} - 16384) / 5215$ This conversion ensures that the maximum allowed value range of $[-\pi, 3*\pi]$ is fully utilized.



Figure 10: Example representation of a beam as a tuple. There are three echoes available. Both distance and RSSI values exist per echo, i.e., the corresponding bits of DataContentEcho have the value 1. Values for beam properties and azimuth angle are also available, i.e., the corresponding bits of DataContentBeams have the value 1.

The individual tuples are located directly behind each other in the data stream. Their sequence is described in figure 11. Here the data is arranged in a matrix where the individual layers of the current module correspond to the rows, and the columns correspond to the measurement beams of the scans for the individual layers. (Note: This arrangement assumes that all scans of a module have the same length, which is ensured by the division into modules as described in section 5.2.) The order of the beam tuples in the data stream is determined by sweeping the matrix shown in figure 11 column by column, i.e., the tuples of beam index 0 for all layers are incorporated into the data stream first, then the tuples for beam index 1, etc.

Module with one single layer



Module with multiple layers

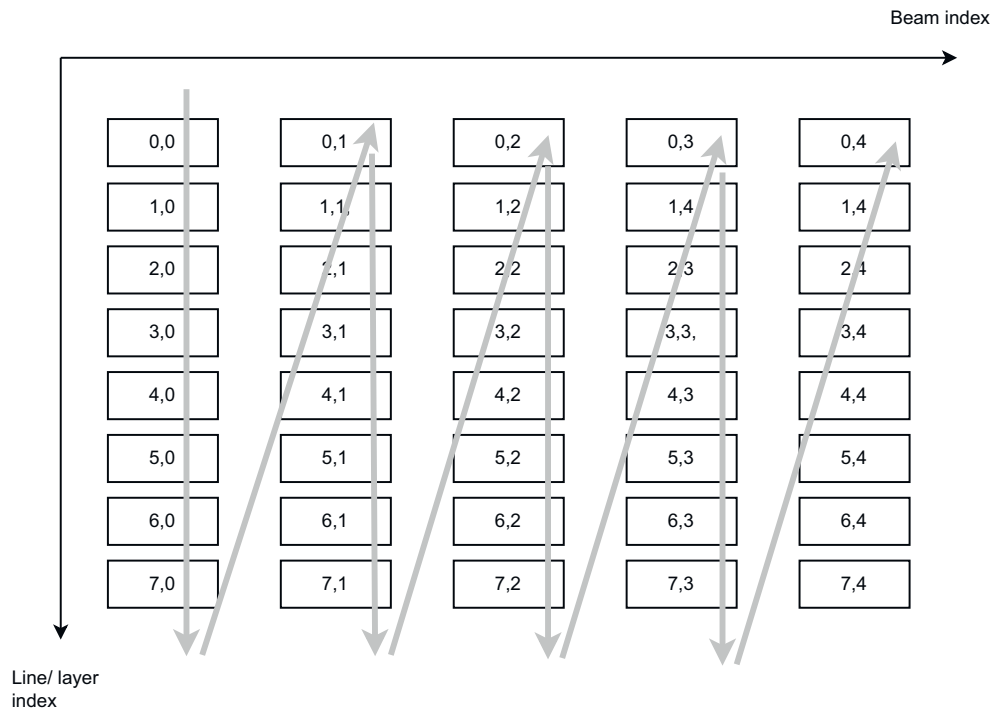


Figure 11: Sequence of data tuples in the memory for the individual beams. Top: Module with only one layer. Bottom: Module with 8 layers. The individual layers of a module correspond to the rows, and the individual measurement beams correspond to the columns. For linear storage of the data in the data package, this matrix is then swept column by column, as indicated by the gray arrows in the figure.

The following correspondences exist between the metadata in table 11 and the measurement data as shown in figure 11:

- The elevation angle of the data in row i is $\text{Phi}[i]$ from the metadata.
- The azimuth angle of the first beam in line i is $\text{ThetaStart}[i]$
- The azimuth angle of the last beam in line i is $\text{ThetaStop}[i]$

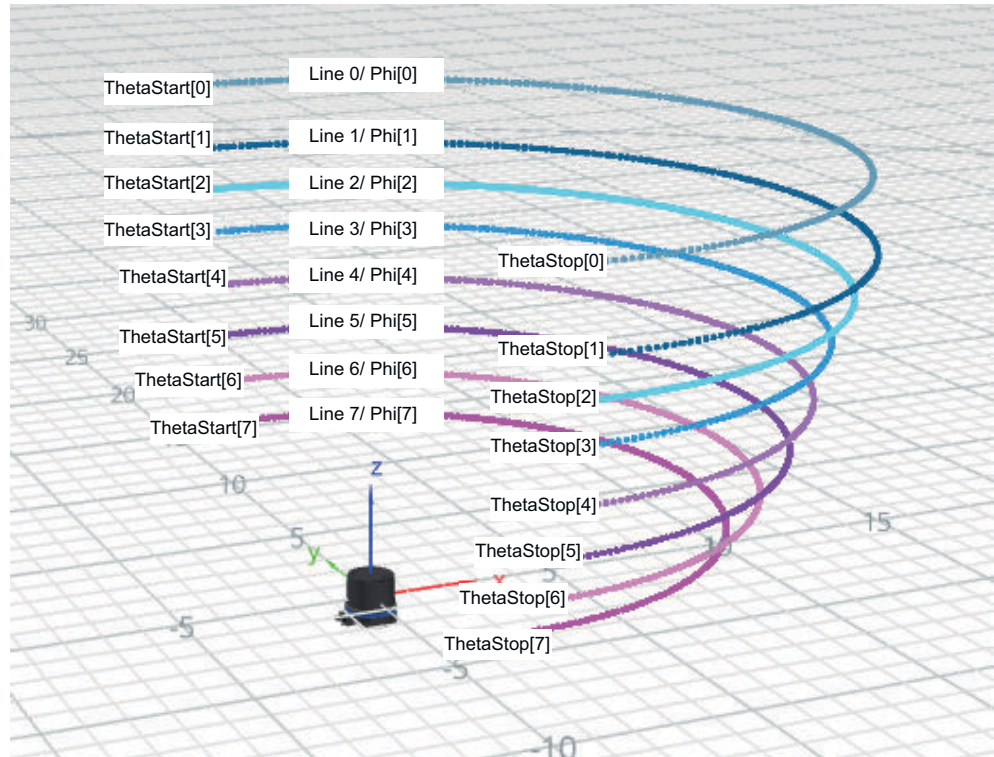


Figure 12: Correspondences between measurement data and metadata using the example of a module with 8 layers. Each line from figure 11 corresponds to a layer in figure 10. The first and last azimuth angles of each layer, and the elevation angles of each layer are in the Phi, ThetaStart and ThetaStop fields.

NOTE | LayerIds, which define the individual layers by their elevation angle as in the MSGPACK format (see table 6, page 10), are not used in the Compact format. The row index of the matrix with the data relates only to the current module and is therefore generally not the same as the LayerId.

5.4 IMU format

The IMU data shall be streamed in the following format:

Word	Value	Size	Data type	Unit	Description
0	Start of Frame	4 bytes	uint32		Always four stx characters: <code>\x2\x2\x2\x2</code> .
1	Command ID	4 bytes	uint32		Defines the type of the transmitted telegram. For IMU data serialization the commandId is 2.
2	Telegram version	4 bytes	uint32		Version of the serialization telegram. For the telegram structure described in this requirement the Telegram version is 1.
3	Acceleration x	4 bytes	float	m/s ²	Acceleration along the x-axis including gravity; gravity is not subtracted from the data.

Word	Value	Size	Data type	Unit	Description
4	Acceleration y	4 bytes	float	m/s ²	Acceleration along the y-axis including gravity; gravity is not subtracted from the data.
5	Acceleration z	4 bytes	float	m/s ²	Acceleration along the z-axis including gravity; gravity is not subtracted from the data.
6	Angular velocity x	4 bytes	float	rad/s	
7	Angular velocity y	4 bytes	float	rad/s	
8	Angular velocity z	4 bytes	float	rad/s	
9	Orientation quaternion w	4 bytes	float	1	
10	Orientation quaternion x	4 bytes	float	1	
11	Orientation quaternion y	4 bytes	float	1	
12	Orientation quaternion z	4 bytes	float	1	
13 14	IMU sensor time stamp	8 bytes	uint64	μs	Sensor system time since 1.1.1970 00:00 in UTC.
15	Check sum	4 bytes	uint32		CRC32 of all words except the checksum.

The word size shall be 32 Bits. All words shall have little endian byte ordering.

The data shall be given in the following coordinate system which is based on the DIN 70000 system: the x-axis lies on the 90° beam of the 0° layer. The y-axis is perpendicular to the x-axis and lies in the 0° layer. The y-values are rising in the counter clock wise rotation direction (right handed system). The z-axis is perpendicular to the x-y-plane and the device top points to rising z-values.



6 Definitions applicable to both data formats

6.1 Azimuth angle

The azimuth angles (also called theta angles in the data formats) of a scan are always monotonically increasing.

For sensors that have a horizontal measuring field of 360° , this means in particular that the angles of a segment can be greater than 180° if the segment exceeds the $+180^\circ/-180^\circ$ limit:

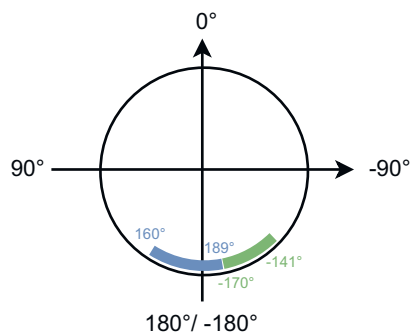


Figure 13: Azimuth angle for different segments. The angular range of the blue segment exceeds the $180^\circ/-180^\circ$ limit. Since the azimuth angles are nevertheless monotonically ascending, angles $> 180^\circ$ are used here. The green segment already starts in the negative angular range, which is why the azimuth angles are negative here.

6.2 Segment counter

The segment counter (SegmentCounter in [table 6](#) and [table 11](#)) counts the segments in a frame. A frame is all the data recorded in one revolution. The segment counter is a value between 0 and $< \text{number of segments per revolution} - 1$ and increases with increasing azimuth angle.

An exception to this is shown in [figure 14](#): Here the beams of a segment exceed the $+180^\circ/-180^\circ$ limit so most azimuth angles of this segment have values $> 180^\circ$, see ["Azimuth angle", page 23](#). (This would correspond to negative values close to -180° when normalized to $(-180^\circ, +180^\circ]$.) If the majority of the values of the segment are $> 180^\circ$, this segment is assigned the segment counter 0 again. This only applies to the multiScan due to the specific arrangement of the lasers in the measuring module.

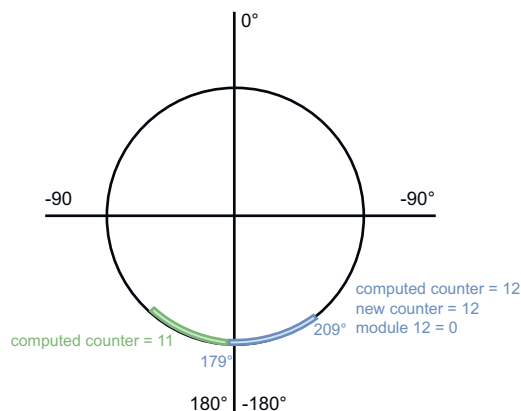


Figure 14: Segment counter for a segment (blue), the majority of whose azimuth angles are $> 180^\circ$. Assuming a segment size of 30° , the segment counter would be 12 if it were calculated only on the basis of the angle values. The majority of the blue segment is already in the next frame, however, which is why it is assigned the segment number 0.

6.3 Representation of RSSI values

RSSI values are represented as a 16-bit integer (uint16). The RSSI is a dimensionless quantity. The values can fall within the complete value range between 0 and $2^{16} - 1$, whereby it is possible that the maximum value is rarely or even never reached. The RSSI is generally also not standardized and therefore not exactly comparable between devices.

6.4 Representation of bit fields

For both the DataContentEchos and DataContentBeams values and for the beam properties (see "Representation of beam characteristics ("Properties")", page 25), bytes are interpreted as bit fields in which information about the states of the individual bits is encoded.

The relationship between the representation of the byte as a decimal value and its bit indices is as follows:

- Let b_i with $i = 0, \dots, k$ be the bits with index i in the binary representation of a value v with, $b_i \in \{0, 1\}$.
- The decimal representation v_{dec} of v is then given as $v_{dec} = \sum_{i=0..7} b_i * 2^i$

Examples:

Let v be an 8-bit value with decimal representation v_{dec} and hexadecimal representation v_{hex} . Let the bit indices of v be b_0 to b_7 .

Table 15: Examples of the relationship between the bit indices and the decimal representation of a value

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	1	1	1	1	1	1	1	1
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
vdec	128	64	32	16	8	4	2	1

Table 16: Examples for the decimal value 128.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	1	0	0	0	0	0	0	0
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
vdec	128							
vhex	0x80							

Table 17: Examples for the decimal value 1.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	0	0	0	0	0	0	0	1
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
vdec	1							
vhex	0x01							

Table 18: Examples for the decimal value 130.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
	1	0	0	0	0	0	1	0
	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
vdec	130							
vhex	0x82							

6.5 Representation of beam characteristics (“Properties”)

The additional characteristics of a beam are called “properties” here. They are encoded in a bit field according to ["Representation of bit fields"](#), page 24. The meaning of the individual bit indices is shown in the following table.

Table 19: Description of the bits of the field for beam characteristics (“Properties”)

Bit index	Content
0	1 if a reflector was detected for any echo on this beam, otherwise 0.
1	Reserved
2	Reserved
3	Reserved
4	Reserved
5	Reserved
6	Reserved
7	Reserved

If the reflector bit (bit with index 0) is set for a beam, it can be assumed that the last echo measured for this beam came from a reflector. It is virtually impossible physically for a reflector to be measured for an echo and then be followed by another on the same beam.

7 Behavior of serialization for data reduction

Different data reduction options (e.g., limiting the number of available echoes, limiting the azimuth angular range, limiting the layers used) affect the data output in different ways.

The behavior is somewhat different for the MSGPACK format and the Compact format. The data reduction can be configured either via SOPASair or via the corresponding telegrams (see www.sick.com/8014631). These effects are described in this section.

7.1 Behavior in relation to the number of available echoes

Multi-echo capable sensors usually provide the option to define the number of available echoes by means of an echo filter. This then affects the data output as follows:

All echoes setting

If the sensor is configured so that all echoes are available, then all theoretically available echoes are also always serialized, regardless of the number of echoes actually received per beam. If fewer echoes than theoretically available are received on a beam, the measured values for the remaining echoes (distance and RSSI) are padded with 0 (see also the example in [table 20](#)).

Single echo setting

If the sensor is configured so that only a single echo is available (e.g., last or first echo), only the data for this echo is serialized.

Table 20: Example data output for different numbers of available echoes.

Angles	All echoes			Last echo
	Distance echo 0	Distance echo 1	Distance echo 2	Distance
42°	3,422 mm	5,022 mm	0 mm	5,022 mm
43°	3,420 mm	0 mm	0 mm	3,420 mm
44°	0 mm	0 mm	0 mm	0 mm

Example:

- The sensor is capable of measuring a maximum of 3 echoes, and it is configured to output all echoes (**All Echoes** columns) or the last echo (**Last Echo** column).
- For the angle 42° the measured distance is 3,422 mm for echo 0 and 5,022mm for echo 1, for the angle 43° the measured distance is 3,420 mm for echo 0, and for angle 44° no valid distance is measured.
- The following distance values are then output (RSSI values are omitted for reasons of clarity, they are treated in the same way as distance values):

7.2 Behavior when restricting the azimuth angular range

If a restricted azimuth angular range is configured for the data output, it is possible for complete segments to lie outside the configured angular range. These segments are not serialized and therefore not transmitted from the device to the client.

For sensors with different physical measuring modules, for example the multiScan1xx, it should be noted that complete segments only lie outside the configured angular range if this applies to the data of both measuring modules. If data from one measuring module only lies within the available angular range, the segment will still be output. The layers of the other measuring module for which the data are outside the available angular range, are then treated as follows:

MSGPACK format

If the data of a layer is completely outside the configured angular range, that layer will not be output, see ["Behavior when reducing the available layers", page 27](#). As soon as at least one beam is within the configured angular range, both the distance value and the RSSI value are set to 0 for the remaining beams of the segment, and the azimuth angle remains unchanged.

Compact format

If the data of a layer are completely outside the configured angular range, then in contrast to the MSGPACK format the distance values and the RSSI values are set to 0 for that layer, and the azimuth value is output unchanged. As soon as at least one beam is within the configured angular range, the procedure is the same as for the MSGPACK format: For the remaining beams of the segment, both the distance value and the RSSI value are set to 0, and the azimuth angle remains unchanged.

The different behavior for the MSGPACK format and the Compact format is due to the fact that in the case of the Compact format, the data structure needs to remain the same for each segment so the data can be easily interpreted (e.g., via memcpy into a structure). This is not necessary with the MSGPACK format, where the data has to be parsed anyway. A more extensive data reduction is therefore possible in this case.

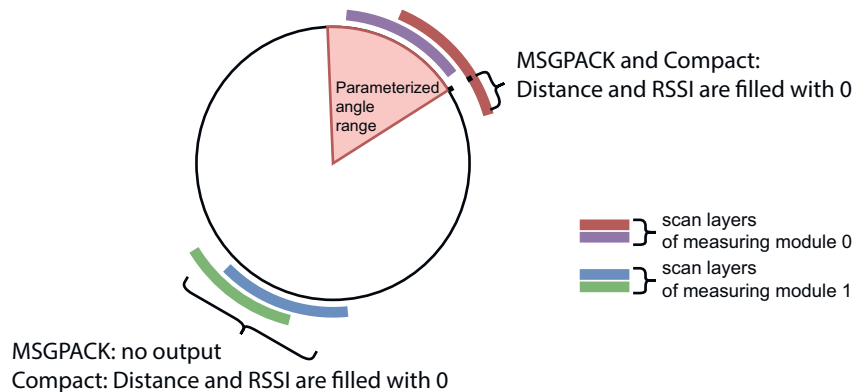


Figure 15: Handling of data that is outside the configured angular range. Data from 4 layers are shown, all belonging to one segment but to different measuring modules. red and violet: measuring module 0, blue and green: measuring module 1. With no restriction of the angular range, the data would be output for all layers. With the angular range restriction shown (red circle segment), the data is output as follows: The blue layer and the green layer are not output in the MSGPACK format, and in the Compact format distance and RSSI are filled with 0. Distance and RSSI values in the part of the red layer outside the configured angular range are padded with 0 in the MSGPACK format and Compact format.

7.3 Behavior when reducing the available layers

For multilayer sensors, for example the multiScan1xx, it is possible for individual layers to fall within the measuring range excluded by data reduction. This can occur, for example, when using a layer filter. The layers for the complete azimuth angular range are excluded. The case where the data for a layer are outside the measuring range only in individual segments is described in ["Behavior when restricting the azimuth angular range", page 26](#).

For the MSGPACK format, this affects the serialization as follows:

- Scan type data for the layers that have been excluded will no longer appear in the SegmentData array (see table 6, page 10).
- The LayerId array (see table 6, page 10) is adjusted accordingly, i.e., the IDs of the layers that are no longer present are removed.

For the Compact format, this affects the serialization as follow:

- The data for the layers that were excluded are removed from the measurement data block, i.e., the corresponding rows of the matrix in [figure 11](#) are removed.
- The metadata in the metadata block that are associated with this line will be removed as well. The arrays affected are ThetaStart, ThetaStop, TimeStampStart, TimeStampStop and Phi in [table 11](#).
- If all layers of a module are outside the configured measuring range, the entire module (metadata and measurement data) is not output.

Australia

Phone +61 (3) 9457 0600
1800 33 48 02 – tollfree
E-Mail sales@sick.com.au

Austria

Phone +43 (0) 2236 62288-0
E-Mail office@sick.at

Belgium/Luxembourg

Phone +32 (0) 2 466 55 66
E-Mail info@sick.be

Brazil

Phone +55 11 3215-4900
E-Mail comercial@sick.com.br

Canada

Phone +1 905.771.1444
E-Mail cs.canada@sick.com

Czech Republic

Phone +420 234 719 500
E-Mail sick@sick.cz

Chile

Phone +56 (2) 2274 7430
E-Mail chile@sick.com

China

Phone +86 20 2882 3600
E-Mail info.china@sick.net.cn

Denmark

Phone +45 45 82 64 00
E-Mail sick@sick.dk

Finland

Phone +358-9-25 15 800
E-Mail sick@sick.fi

France

Phone +33 1 64 62 35 00
E-Mail info@sick.fr

Germany

Phone +49 (0) 2 11 53 010
E-Mail info@sick.de

Greece

Phone +30 210 6825100
E-Mail office@sick.com.gr

Hong Kong

Phone +852 2153 6300
E-Mail ghk@sick.com.hk

Hungary

Phone +36 1 371 2680
E-Mail erteakesites@sick.hu

India

Phone +91-22-6119 8900
E-Mail info@sick-india.com

Israel

Phone +972 97110 11
E-Mail info@sick-sensors.com

Italy

Phone +39 02 27 43 41
E-Mail info@sick.it

Japan

Phone +81 3 5309 2112
E-Mail support@sick.jp

Malaysia

Phone +603-8080 7425
E-Mail enquiry.my@sick.com

Mexico

Phone +52 (472) 748 9451
E-Mail mexico@sick.com

Netherlands

Phone +31 (0) 30 204 40 00
E-Mail info@sick.nl

New Zealand

Phone +64 9 415 0459
0800 222 278 – tollfree
E-Mail sales@sick.co.nz

Norway

Phone +47 67 81 50 00
E-Mail sick@sick.no

Poland

Phone +48 22 539 41 00
E-Mail info@sick.pl

Romania

Phone +40 356-17 11 20
E-Mail office@sick.ro

Singapore

Phone +65 6744 3732
E-Mail sales.gsg@sick.com

Slovakia

Phone +421 482 901 201
E-Mail mail@sick-sk.sk

Slovenia

Phone +386 591 78849
E-Mail office@sick.si

South Africa

Phone +27 10 060 0550
E-Mail info@sickautomation.co.za

South Korea

Phone +82 2 786 6321/4
E-Mail infokorea@sick.com

Spain

Phone +34 93 480 31 00
E-Mail info@sick.es

Sweden

Phone +46 10 110 10 00
E-Mail info@sick.se

Switzerland

Phone +41 41 619 29 39
E-Mail contact@sick.ch

Taiwan

Phone +886-2-2375-6288
E-Mail sales@sick.com.tw

Thailand

Phone +66 2 645 0009
E-Mail marcom.th@sick.com

Turkey

Phone +90 (216) 528 50 00
E-Mail info@sick.com.tr

United Arab Emirates

Phone +971 (0) 4 88 65 878
E-Mail contact@sick.ae

United Kingdom

Phone +44 (0)17278 31121
E-Mail info@sick.co.uk

USA

Phone +1 800.325.7425
E-Mail info@sick.com

Vietnam

Phone +65 6744 3732
E-Mail sales.gsg@sick.com

Detailed addresses and further locations at www.sick.com