Scanse™

User's Manual and Technical Specifications

sweep **v1.0**
scanning laser range finder

- ✓ Cost efficient design
- ✓ Operates in full sunlight
- ✓ Low power consumption
- ✓ Wide field of view
- ✓ Small footprint
- ✓ Simple serial connectivity
- ✓ Long Range

⚠ CAUTION

Laser Safety

This device contains a component which emits laser radiation. The laser product is designated Class 1 during all operating modes. This means that the laser is safe to look at with the unaided eye, however it is advisable to not look directly into the beam when in use.

Power Safety

When connecting a Sweep sensor to a 5VDC power source, it should be limited to a maximum of 8A as defined in EN 60950-1, sub clause 2.5, Table 2B.

Documentation Revision Information

Rev	Date	Changes
0.9	12/19/2016	Initial Release
0.91	01/05/2016	Added MS, LR, LI Packets
0.92	01/06/2017	Added power safety text
0.93	01/13/2016	Added LED status and Pin details
0.94	01/24/2016	Corrected typos and added Appendix

Table of Contents

Specifications

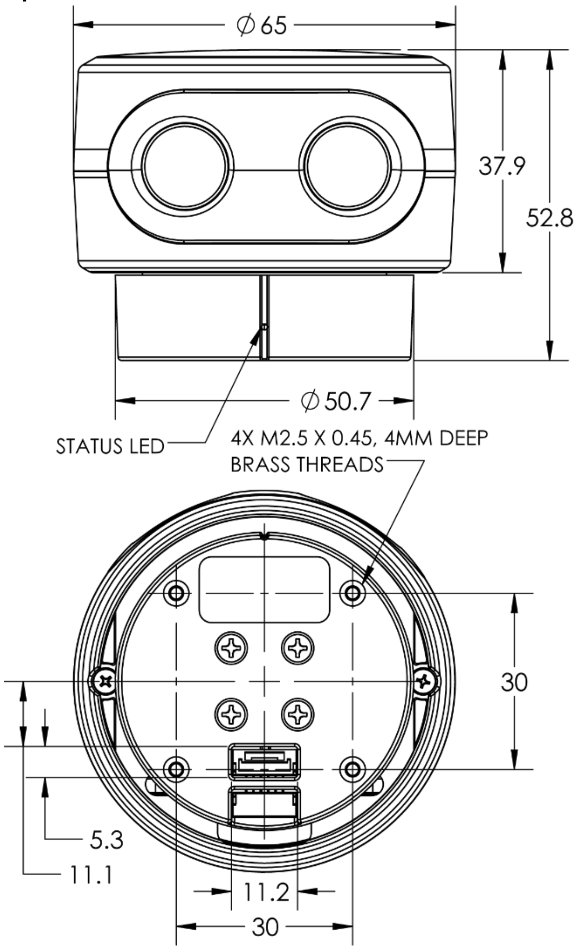

∅ 65
37.9
52.8
∅ 50.7
STATUS LED
4X M2.5 X 0.45, 4MM DEEP BRASS THREADS
30
5.3
11.1
11.2
30
ALL DIMENSIONS ARE IN MM, DRAWINGS ARE NOT TO SCALE

Figure 1, Sweep Dimension Drawing

Physical

Specification	Value
Weight	120 g (4.23 oz.)
Operating Temperature	-10 to 60° C (14 to 140°F)
Storage Temperature	-40 to 80° C (-40 to 176°F)

Electrical

Specification	Value
Power	5VDC ±0.5VDC
Current Consumption	Up to 650mA 450mA nominal

Measurement Performance

Specification	Value
Range (75% reflective target)	40 m (131ft)
Resolution	1 cm (0.4 in)
Update Rate (75% reflective target)	Up to 1075Hz (see "Theory of Operation")

Field of View

Specification	Value
Horizontal Field of View	360 degrees
Vertical Field of View	0.5 degrees

Sweep is a single plane scanner. This means that as its head rotates **counterclockwise**, it records data in a single plane. The beam starts out at approximately 12.7mm in diameter and expands by approximately 0.5 degrees as show in Figure 2.

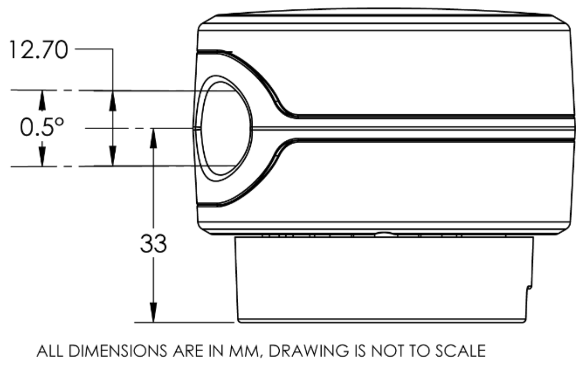
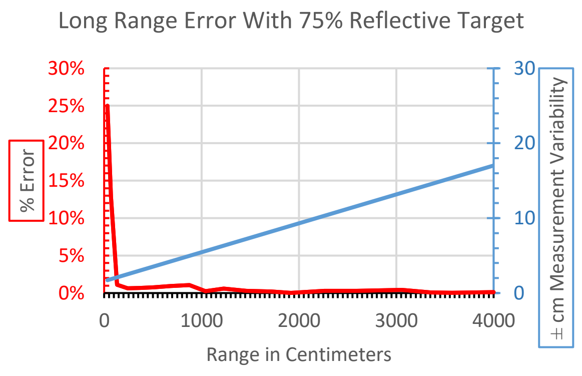


Figure 2, Sweep Field of View

Measurement Error Test Data



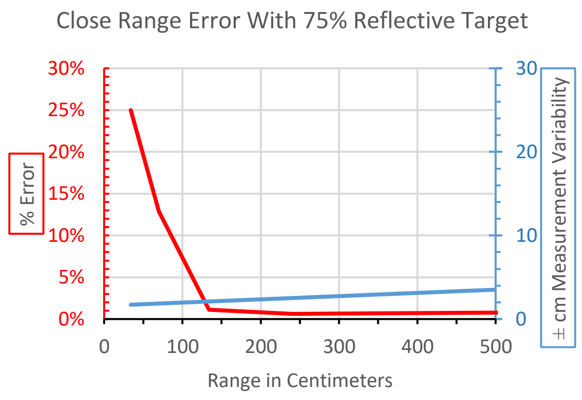


Figure 3, Sweep Accuracy Graphs

Overview of Interfaces

Sweep can be connected to low level micro controllers directly using its serial port, or to a PC using the provided USB to serial converter.

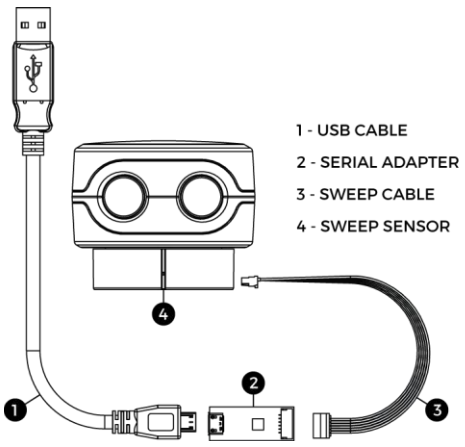
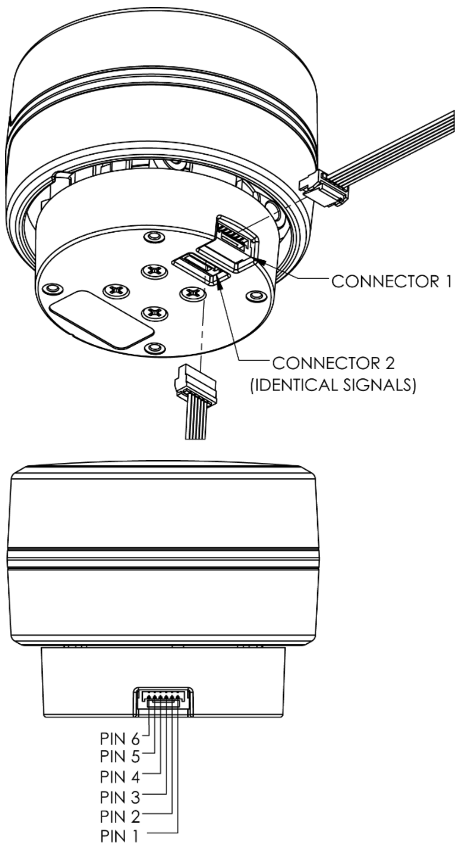


Figure 4, Sweep Cable Diagram

Connector

Sweep has two serial port connectors with identical signals. This allows for more mounting options.



DRAWINGS ARE NOT TO SCALE

Figure 5, Sweep Connector Diagram

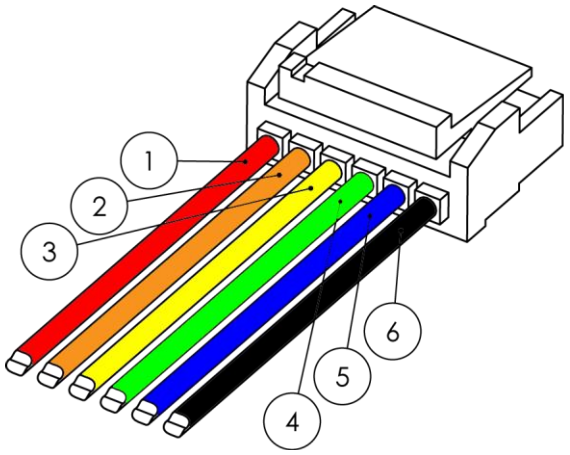


Figure 6, Sweep Pigtail Cable Connector Detail

Table 1, Pin Definition

Pin	Color	Function
1	Red	5VDC ±0.5VDC
2	Orange	Power enable (internal pull-up). Pull down to put device in sleep mode.
3	Yellow	Sync/Device Ready Goes high when first range measurement of new scan is completed, then goes low when second range measurement is completed. Remains low until next rotation. Line is only active when scanning and low when not.
4	Green	UART RX 3.3V (5V compatible)
5	Blue	UART TX 3.3V (5V compatible)
6	Black	Ground (-)

You can create your own cable if needed for your application. These components are readily available.

Part	Description	Mfg.	Part No.
Connector Housing	6-Position, rectangular housing, latch-lock connector receptacle with 1.25 mm (0.049 in.) pitch.	JST	GHR-06V-S
Connector terminal	26-30 AWG crimp socket connector	JST	SSHL-002T-P0.2
Wire	UL 1061 26 AWG stranded copper	N/A	N/A

Status LED

The status LED (Light Emitting Diode) located on the base of the sensor can give valuable feedback.

Display	Meaning
Blinking Green	Initial startup routine. No data is output at this time.
Solid Blue	Normal operation.
Solid Red	Internal communication error.

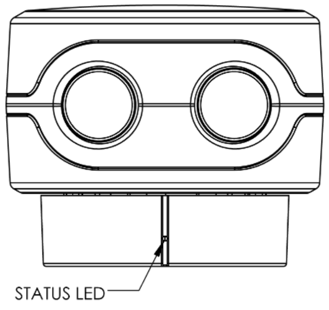


Figure 7, Status LED Location

Mounting and Vibration Considerations

Sweep can be mounted in any orientation. Sweep's rotating head is dynamically balanced, which means it is immune to linear vibration, but it can be affected by rotational vibration. Sudden rotational shocks can cause the head to either slow down or speed up, which can affect data accuracy. If Sweep is rotationally jerked hard enough, it can cause the motor to lose sync, which will trigger a momentary motor pause, and then restart.

Mounting Features and Orientation

Sweep has four brass threaded inserts designed to fit M2.5X0.45 screws in its base. These threaded holes are the way to mount Sweep to your device. The threaded holes are aligned with the scanner's measurement angles. The scanner's zero degree measurement starting angle is aligned with the status LED, as shown in Figure 8.

Ingress Protection Rating

Sweep is rated as IP51, which is to say, it is not dust or water tight. **It is recommended that Sweep be placed inside a protective transparent enclosure if it will be used in dusty or wet environments.**

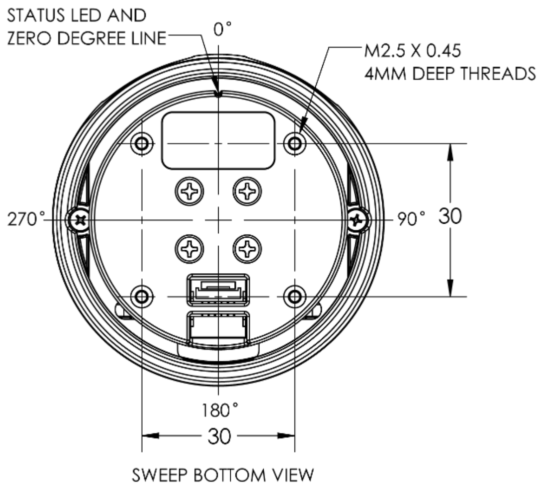


Figure 8, Sweep Mounting Features (all dimensions in mm)

Enclosure Window Design

Sweep uses 905nm laser light, which passes through several kinds of clear glass and plastic very well. Based on our testing, clear Polycarbonate plastic is one of the best choices, as it can be molded to fit the profile of the application's enclosure, is very inexpensive, and in most cases, is more than 95% translucent to Sweep's light beam. Factors that can affect the performance of a window are:

- Thickness of the window. Thicker windows will block more light, as well as bend the light more if the beam is not hitting the window normal to the surface.
- Scratches and dust. The presence of scratches and dust on the window will scatter the laser light, and may reflect some of the light back into the sensor's detector, causing measurement errors.
- Surface coatings. There are a variety of coatings that can help with the performance of windows. One is an anti-reflective (AR) coating, which can help reduce the amount of laser light that is reflected as it passes through the window's surface.

Theory of Operation

Distance Measurement

Sweep employs a time of flight ranging method. This technique involves transmitting a packet of micro pulses of light in a unique pattern. When this light bounces off an object and returns to the receiving detector, a correlation algorithm is used to identify the unique light pattern from ambient noise. **Each light packet is different from the last, which allows multiple Sweep sensors to operate adjacent to each other without interference**.

The light packets that Sweep uses can vary in length, which can affect accuracy of range measurements, as well as the maximum range and update rate. Under normal operation, Sweep limits the maximum time per measurement to a value determined by the sample rate set using the **LR** command (see LR packet structure description). If not enough light is returned from the environment, the measurement fails, and a 1 is returned as the range value. On the other hand, if a lot of light is returned from the environment, the correlation algorithm can reach its maximum accuracy early, and can return a range value more quickly. This is what makes the update rate of Sweep variable. The value of setting a slower sample rate using the **LR** command, is that more light will be gathered from a target, and the range measurements will be more accurate. The exact accuracy is determined by many factors, including the target surface characteristics and ambient noise, so we cannot give an exact number for relative accuracy between the different **LR** settings.

Angle Measurement

Sweep uses an optical encoder to measure the angle of the rotating sensor head. The angle that is recorded for a range data point is the angle the sensor is at when the measurement is completed. The beginning of the scan, and zero degrees is located where the status LED project out of the base of the sensor, as indicated in Figure 7.

Applications

Sweep can be used for a variety of applications, including robot guidance/obstacle avoidance, 3D scanning, surveying, people tracking and many more.

Internal Filters

Sweep has the ability to perform some simple data filtering within the sensor itself. These filters are still in development, and are being made for specific customer segments. Examples include having Sweep split up its field of view into eight sections, then transmit only the closest objects within each of those sectors. Another example is to have Sweep only output data from a range of angles. **If you have an application that requires a specific filter, please contact us.**

Visualizer Overview

You can download the Sweep visualizer at www.scanse.io/downloads
The purpose of the Scanse visualizer is to provide a way to quickly evaluate Sweep's performance in your application/environment. For some applications, like surveying, our visualizer can be used to take quick measurements between range data points within a scan. It contains a programming tool for updating Sweep's firmware.

A full tutorial for using the visualizer can be found in software support section at support.scanse.io.

Serial Protocol Specification

Specification	Value
Bit Rate	115.2 Kbps
Parity	None
Data Bit	8
Stop Bit	1
Flow Control	None

Data Encoding and Decoding

All characters used for commands and responses are ASCII code in addition to CR and LF, except for the measurement packet.

Communication Format

All communication packets between the host computer and the sensor begin with ASCII letter command codes.

Available Command Codes

ASCII Code (2 bytes)	Function
DS	Start data acquisition
DX	Stop data acquisition
MS	Adjust Motor Speed
LR	Adjust LiDAR Sample Rate
MI	Motor Information
IV	Version Info
ID	Device Info
RR	Reset Device

General Communication Packet Structure

(HOST -> SENSOR)

Command with no parameter

Command Symbol (2 bytes)	Line Feed (LF)
Example: DS, DX, MI, IV…	Line Feed (LF) or Carriage Return (CR)

or

Command with parameter

Command Symbol (2 bytes)	Parameter (2 bytes)	Line Feed (LF)
Example: MS, LR…	2 bytes	Line Feed (LF) or Carriage Return (CR)

(SENSOR -> HOST)

Response with no parameter echoed

Command Symbol (2 bytes)	Status (2 bytes)	Sum of Status	Line Feed (LF)

or

Command with parameter echoed

Command Symbol (2 bytes)	Parameter (2 bytes)	Line Feed (LF)	Status (2 bytes)	Sum of Status	Line Feed (LF)

Definition of terms:

Command Symbol: 2 byte code at the beginning of every command (ASCII)

Parameter: Information that is needed to change sensor settings (ASCII).. Example: a motor speed code 05 transmitted as ASCII parameter '05', which has byte values [48, 53] in decimal.

Line Feed (LF) or Carriage Return (CR): Terminating code. Command can have LF or CR or both as termination code but receipt will always have LF as its termination code.

Status: 2 bytes of data used to convey the normal/abnormal processing of a command. ASCII byte values of '00' or '99' indicate that the sensor received and processed the command normally. Value of '11' specifies an invalid parameter was included in the command. Any other byte values are reserved for errors, which will convey that an undefined, invalid or incomplete command was received by the sensor.

Sum of Status: 1 byte of data used to check for corrupted transmission. See Appendix for instructions for authenticating receipts.

DS - Start data acquisition

- Initiates scanning
- Responds with header containing status.
- Next responds with measurement packets indefinitely until commanded to stop.

(HOST -> SENSOR)

D	S	LF

(SENSOR -> HOST)

Header response

D	S	Status (2 bytes)	SUM (2 bytes)	LF

Data Block (7 bytes) Data Block

Sync/Error (1 byte)	Azimuth - degrees(float) (2 bytes)	Distance - cm(int) (2 bytes)	Signal Strength (1 byte)	Checksum (1 byte)

Data Block Structure:

The Data Block receipt is 7 bytes long and contains all the information about a single sensor reading.

- **Sync/Error Byte:** The sync/error byte is multi-purpose, and encodes information about the rotation of the Sweep sensor, as well as any error information. Consider the individual bits:

e6	e5	e4	e3	e2	e1	e0	sync

  - **Sync bit**: least significant bit (LSB) which carries the sync value. A value of 1 indicates that this Data Block is the first acquired sensor reading since the sensor passed the 0 degree mark. Value of 0 indicates all other measurement packets.
  - **Error bits**: 7 most significant bits (e0-6) are reserved for error encoding. The bit e0 indicates a communication error with the LiDAR module with the value 1. Bits e1:6 are reserved for future use.
- **Azimuth:** Angle that ranging was recorded at (in degrees). Azimuth is a float value, transmitted as a 16 bit int. This needs to be converted from 16bit int to float. Use instructions in the Appendix. Note: the lower order byte is received first, higher order byte is received second.
- **Distance:** Distance of range measurement (in cm). Distance is a 16 bit integer value. Note: the lower order byte is received first, higher order byte is received second. Use instructions in the Appendix.
- **Signal strength :** Signal strength of current ranging measurement. Larger is better. 8-bit unsigned int, range: 0-255
- **Checksum:** Calculated by adding the 6 bytes of data then dividing by 255 and keeping the remainder. (Sum of bytes 0-5) % 255 ... Use the instructions in the Appendix.

DX - Stop data acquisition

Stops outputting measurement data.

(HOST -> SENSOR)

D	X	LF

(SENSOR -> HOST)

D	X	Status	SUM	LF

MS – Adjust Motor Speed

Once a speed is set, the sensor will always return to this speed, even after a power cycle (except when setting the speed to 0Hz – in which case it will go back to 5Hz after a power cycle).

(HOST -> SENSOR)

M	S	Speed Parameter (2 bytes)	LF

Speed Parameter: in ASCII 00 - 10 : 10 different speed levels according to Hz, increments of 1. ie: 01,02,..
00 = Motor stopped.

(SENSOR -> HOST)

M	S	Speed(Hz) (2 bytes)	LF	Status	Sum	LF

LR – Adjust LiDAR Sample Rate

Default Sample Rate: 500-600Hz. See Theory of Operation section for explanation of why there is a range of sample rate values.

(HOST -> SENSOR)

L	R	Speed Parameter (2 bytes)	LF

Sample Rate Parameter Code in ASCII:
01 = 500-600Hz
02 = 750-800Hz
03 = 1000-1075Hz

(SENSOR -> HOST)

L	R	Speed(Hz) (2 bytes)	LF	Status	Sum	LF

LI – LiDAR Information

Returns current LiDAR Sample Rate Code in ASCII:
01 = 500-600Hz
02 = 750-800Hz
03 = 1000-1050Hz

(HOST -> SENSOR)

L	I	LF

(SENSOR -> HOST)

L	I	Speed(Hz) (2 bytes)	LF

MI – Motor Information

Returns current rotation frequency in Hz in ASCII 00 - 10 (increments of 1)

(HOST -> SENSOR)

M	I	LF

(SENSOR -> HOST)

M	I	Speed(Hz) (2 bytes)	LF

IV - Version Details

- Model
- Protocol Version
- Firmware Version
- Hardware Version
- Serial Number

(HOST -> SENSOR)

I	V	LF

(SENSOR -> HOST)

I	V	Model (5 bytes)	Protocol (2 bytes)	Firmware Version (2 bytes)	Hardware Version (1 byte)	Serial Number (8 bytes)	LF
Example: IVSWEEP01011100000001							
I	V	SWEEP	01	01	1	00000001	LF

ID - Device Info

- Bit Rate
- Laser State
- Mode
- Diagnostic
- Motor Speed
- Sample Rate

(HOST -> SENSOR)

I	D	LF

(SENSOR -> HOST)

I	D	Bit Rate (6 bytes)	Laser state	Mode	Diagnostic	Motor Speed (2 bytes)	Sample Rate (4 bytes)	LF
Example: IV115200110050500								
I	D	115200	1	1	0	05	0500	LF

RR - Reset Device

- Reset Scanner

(HOST -> SENSOR)

R	R	LF

(SENSOR -> HOST)

No Response

Appendix:

Authenticating Receipts (Does not apply to Data Block)

Authentication of a valid receipt is accomplished by a checksum, which uses the Status (2 bytes) and Sum of Status (1 byte) from the receipt. To perform the checksum, the received Sum of Status bytes is checked against a calculated sum of the 2 Status bytes. The protocol design allows for performing the checksum visually in a terminal, which requires all bytes in a receipt to be legible ASCII values (ex: '00P'). Therefore, performing the checksum in code is not intuitive. It works like this:

- The status bytes are summed
- The lower 6 bits (Least Significant) of that sum are then added to 30Hex
- The resultant value is compared with the checksum byte

```
//statusByte#1 + statusByte#2
let sumOfStatusBytes = status1_byteValue + status2_byteValue;
//grab the lower (least significant) 6 bits by performing a bit-wise AND with 0x3F (ie: 00111111)
let lowerSixBits = sumOfStatusBytes & 0x3F;
//add 30Hex to it
let sum = lowerSixBits + 0x30;
return ( sum === checkSumByteValue );
```

Example: Consider the common case of `'00P'` (decimal -> `[48, 48, 80]`, hex -> `[0x30, 0x30, 0x50]`)

```
0x30 + 0x30 = 0x60 // sum of the status bytes
0x60 & 0x3F = 0x20 // retrieve only the lower 6 bits
0x20 + 0x30 = 0x50 // calculate the ASCII legible sum
0x50 = 'P'         // translate to ASCII
```

Parsing Data Block 16-bit integers and floats

The Data Block receipt includes int-16 and float values (distance & azimuth). In the case of distance, the value is a 16-bit integer. In the case of the azimuth, the value is a float which is sent as a 16bit integer and must be converted back to a float.
In either case, the 16-bit int is sent as two individual bytes. The lower order byte is received first, and the higher order byte is received second. For example, parsing the distance:

```
//assume dataBlock holds the DATA_BLOCK byte array
//such that indices 3 & 4 correspond to the two distance bytes
let distance = (dataBlock[4] << 8) + (dataBlock[3]);
```

For floats, start by using the same technique to acquire the 16-bit int. Once you have it, you can perform the conversion to float like so:

```
//assume dataBlock holds the DATA_BLOCK byte array,
//such that indices 1 & 2 correspond to the two azimuth bytes
let angle_int = (dataBlock[2] << 8) + (dataBlock[1]);
let degree = 1.0 * ( (angle_int >> 4) + ( (angle_int & 15) / 16.0 ) );
```

Performing Data Block Checksum

The last byte of a Data Block receipt is a checksum byte. It represents the sum of all bytes up until the checksum byte (ie: the first 6 bytes). Obviously, a single byte caps at 255, so the checksum can't reliably hold the sum of 6 other bytes. Instead, the checksum byte uses the modulo operation and effectively contains the remainder after dividing the sum by 255 (this is the same as saying sum % 255). Validating the checksum looks like:

```
//ONLY applies to receipts of type ReceiptEnum.DATA_BLOCK
//calculate the sum of the specified status or msg bytes
let calculatedSum = 0;
for(let i = 0; i < 6; i++){
    //add each status byte to the running sum
    calculatedSum += dataBlock[i];
}
calculatedSum = calculatedSum % 255;
let expectedSumVal = dataBlock[6];
return calculatedSum === expectedSumVal;
```