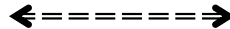


# MAVOSPEC **BASE** SDK

3.0/01.21



## Table of Contents

1	Introduction.....	4
2	Quick Start.....	5
2.1	ZIP-Archive contents .....	5
2.2	Usage of the Excel Sheets .....	5
2.3	Usage of the terminal program (HID) .....	6
3	Protocols .....	8
3.1	HID.....	8
3.1.1	Receiving Data.....	8
3.1.2	Sending Data .....	10
3.2	CDC Protocol (Serial Connection) .....	11
4	Command Descriptions .....	13
4.1	Read Commands .....	13
4.1.1	Get Device Information.....	14
4.1.2	Get Sensor Information .....	14
4.1.3	Get Firmware Revision.....	15
4.1.4	Get Hardware Revision .....	15
4.1.5	Get Wavelength Calibration Coefficients .....	16
4.1.6	Get Calibration Data.....	16
4.1.7	Get Sensor Sensitivity Data.....	17
4.1.8	Get Wavelength (Pixel) .....	17
4.1.9	Get IntegrationTime ms.....	18
4.1.10	Get the Status Flags .....	18
4.1.11	Get Raw spectral data.....	19
4.1.12	Get battery voltage.....	19
4.1.13	Get sensor temperature .....	20
4.1.14	Get single Ambient Sensor Conversion .....	21
4.1.15	Start a Flicker Measurement (needed before 0x24 and 0x25) .....	21
4.1.16	Get all Ambient Sensor Data (needs 0x23 to be done) .....	22
4.1.17	Get calculated flicker values.....	22

4.1.18	Get Flicker Frequency .....	23
4.1.19	Get Calculated Data .....	24
4.1.20	Get currently activated Report Items .....	26
4.1.21	Get Device Time and Date .....	26
4.1.22	Get currently Displayed Function.....	27
4.1.23	Get current Backlite Setting.....	27
4.2	Write Commands.....	28
4.2.1	Set activate Report Items .....	29
4.2.2	Set Device Time and Date .....	29
4.2.3	Send Keystroke.....	30
4.2.4	Set Function to be displayed .....	30
4.2.5	Set Display Brightness .....	31
4.2.6	Set Keylock .....	31
4.2.7	Save Settings in EEPROM .....	32
4.2.8	Beep .....	32
4.3	Example: How to start a Measurement and get the Data .....	33
Appendix A	Possible Transfer Statuses and Error Codes .....	34
Appendix B	Data Structures.....	35
Document Revision History .....		45

## 1 Introduction

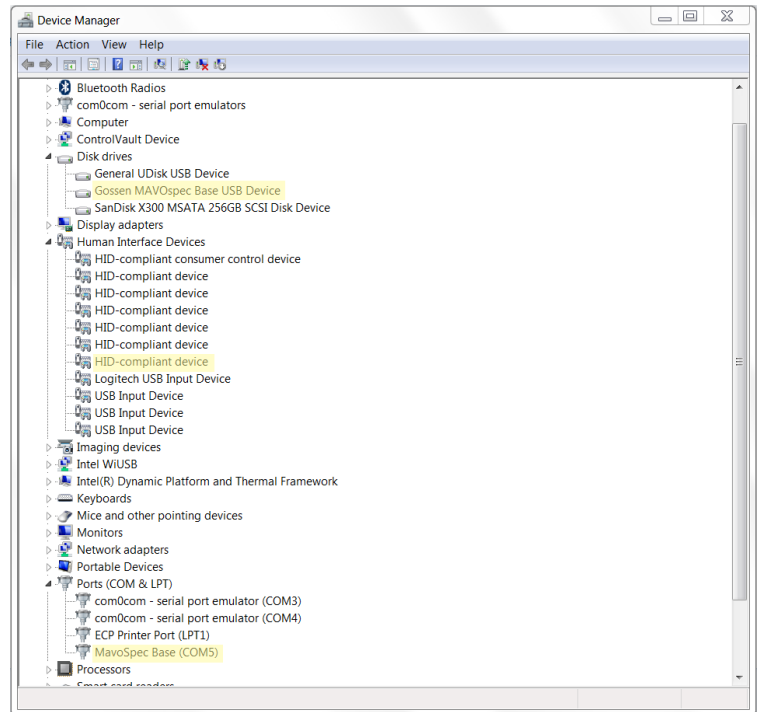
The USB-Interface of the GOSSEN MAVOSPEC **BASE** is a so called “USB composite device”. Meaning, that the device provides more than one interface functions with one connection.

If the device is connected to your computer it will register as a disc drive as well as a HID (Human Interface Device) and CDC (Communications Device Class aka Serial Connection). The disc drive is for loading the saved measurements and the HID and CDC for remote controlling the device and getting measured data.

For Windows 10 no special drivers are needed as all necessary drivers are included in your operating system and are installed automatically when the device is connected to the PC for the first time.

**If you are running Windows 8.1 / 8 or 7 you will need a driver to use the CDC-Interface (Serial Port). You can download this driver (\*.inf – File) from our Homepage.**

If you are still running Microsoft XP (you poor lad), the disc drive and the HID-Device should work. The CDC might(?) work if you find drivers for it (we will not provide them, sorry).



Picture 1: Screenshot of the Device Manager with a connected MAVOSPEC **Base**

## 2 Quick Start

### 2.1 ZIP-Archive contents

In this ZIP-Archive you find the following files:

- MAVOSpec Base SDK Manual.pdf
- MAVOSpec Base HID demo Vxxx.xlsm
- MAVOSpec Base CDC demo Vxxx.xlsm
- MspecHidDemo Vxxx.exe
- MavoSpec Base CDC Driver.inf

this file  
usage example in MS Excel (for HID Protocol)  
usage example in MS Excel (for CDC Protocol)  
small terminal program for the use with a **MAVOSPEC BASE**  
the INF-File for Windows 7 / 8 / 8.1 (CDC-Driver)

If you are missing a file please download the archive again from [https://gossen-photo.de/downloads\\_spektrometer/](https://gossen-photo.de/downloads_spektrometer/) and if the problem persists please contact us via email to [info@gossen-photo.de](mailto:info@gossen-photo.de).

### 2.2 Usage of the Excel Sheets

The Excel Sheet for the HID connection was designed for 32bit Office for Windows. If used on 64bit Office, Excel will throw some errors and it will not work. For full compatibility with Office 32bit and 64bit you can use the CDC Excel Sheet.

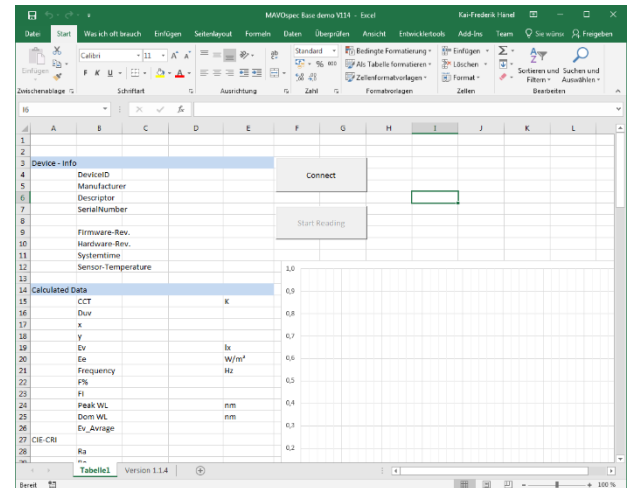
The functionality is the same with HID and CDC Excel Sheets.

Some of the used functions are not available in Excel for MAC so it will not work on a MAC.

In this Sheet are Buttons for connecting to the **MAVOSPEC BASE** ("Connect") and for taking measurements ("Start reading").

After connecting to a device, you can take a measurement. The measured values will be displayed on the first Sheet.

All VBA Macros and Functions are available for reading and editing and are commented.



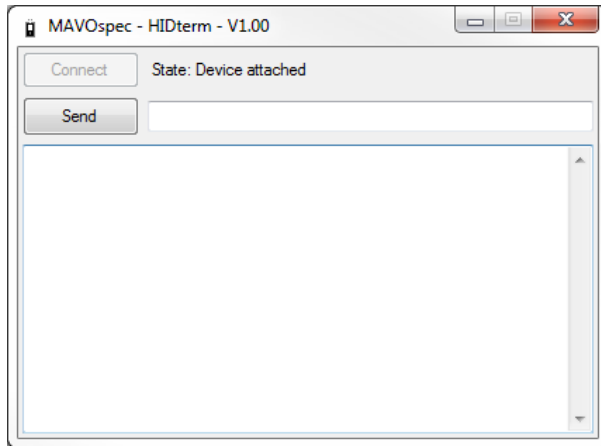
Picture 2: Excel Sheet for taking measurements

### 2.3 Usage of the terminal program (HID)

With the terminal program, "MspecHidDemo V100.exe" you can quick-test the interface off the MAVOSPEC **BASE**.

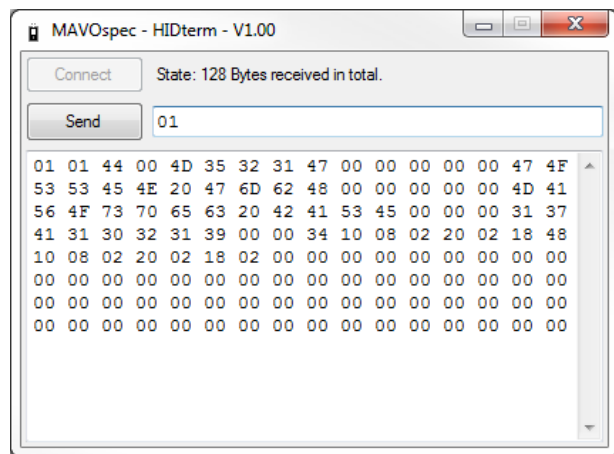
For this application to run, you need "Microsoft .NET Framework Version 3.5" or newer. Since Windows 7 this comes pre-installed with the OS. If it is not installed on your Computer, you can download it directly from Microsoft: <https://www.microsoft.com/en-us/search/result.aspx?q=net+framework>.

To communicate with the MAVOSPEC **BASE** the program has to connect to it first. By pressing the "Connect" button the terminal program searches the connected HID-USB-Devices for a device with the VID = 1CD7 and PID = 4001 (MAVOSPEC **BASE**). If the device is found, it will try to connect to it. If the connection was successful the program will state that the device is connected and the "Send" button is activated.



Picture 3: Terminal program after connecting to the MAVOSPEC **BASE**

Type "01" in the line besides the "Send" button and click "Send" (or press ENTER) to send the command to the MAVOSPEC **BASE**. You can see the response of the device to the command in the output window.



Picture 4: HEX-Dump of the answer to the command "01" (CMD\_GET\_DEVICE\_INFO)

To clear the output window double click in it.

For most of the commands it is enough to send a short (1 Byte) command to the device to get the desired answer. More on that topic will be shown in the following description of the communication-protocol.

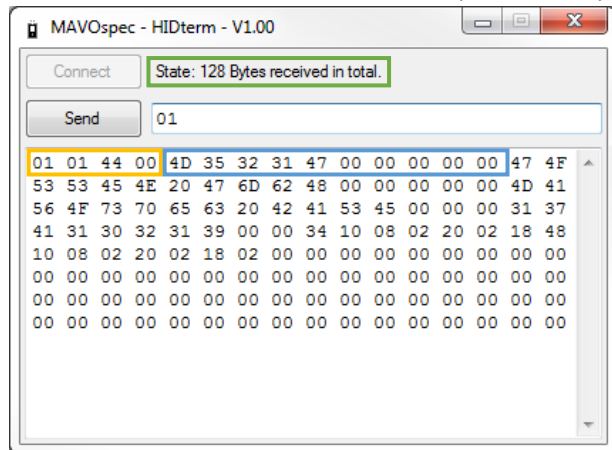
All command sequences, explained in this document, are a description of the "Low Level Protocol" (LLP), which directly triggers actions in the device and returns the results. This protocol grants a quick access to all device functions. But the usage of LLP requires some understanding of the device functions and more programming work to extract / convert the wanted information from the HEX-Data.

### 3 Protocols

#### 3.1 HID

##### 3.1.1 Receiving Data

The command sent before, „CMD\_GET\_DEVICE\_INFO = 0x01“, is the request for the device information. The structure of the LLP will be explained with the received data from the device from the previous example.



Picture 5: Structure of a Low-level telegram

The structure of the sent and received data is fixed, consisting of header and body with the requested data. The HID protocol has a maximum length of 65 Byte per transmission. If the requested data is more than 65 Byte it has to be split into packages of 64 Byte<sup>1</sup> with an additional control byte, the package index (always 0x00).

The **status line** displays, that the received telegram consists of 128 Byte (two packages, 64 Byte each). Data will always be sent in 64 Byte packages for performance reasons. Unused bytes at the end of the telegram will be filled with 0x00.

<sup>1</sup> Remark: The package ID is hidden in the output window. The package ID is also not counted in the length of the payload.



The **telegram header** consists of 4 Bytes:

- |                     |                             |           |                       |
|---------------------|-----------------------------|-----------|-----------------------|
| - Command Code      | 1 Byte                      | 0x01      | = CMD_GET_DEVICE_INFO |
| - Transfer Status   | 1 Byte                      | 0x01      | = HID_DONE            |
| - Length of Payload | 2 Byte (LB HB) <sup>2</sup> | 0x44 0x00 | = 68 Byte Payload     |

The telegram header is transmitted once with the first telegram. The receiving software has to continue to read packages till the whole payload has been received.

The payload is transmitted for every command in the predefined data structure. If the payload has no defined structure, it is transmitted as an array. The array elements are transmitted in little-endian format.

An easy way is, to create these structures in the user-program and fill them with the HEX-Data. It might be necessary to convert data (e.g. for date format) depending on the used programming language.

E.g. the data structure for the command CMD\_GET\_DEVICE\_INFO has following structure:

```
typedef struct {  
    char    DeviceID[10];           // M521G  
    char    Manufacturer[16];       // GOSSEN GmbH  
    char    DeviceDescriptor[16];   // MAVOSpec BASE  
    char    SerialNumber[10];       // 16C10018  
    BCDTIME_T ProductionDate;       // 18.07.2016  
    BCDTIME_T LastServiceDate;      // 18.07.2016  
    PROD_STATE_T ProdState;         // 2  
} DEVICE_INFO_T
```

Table 1: Data structure of the device information

If you compare the **marked text** from the output window (DeviceID = String with length 10),

HEX	4D	35	32	31	47	00	00	00	00
ASCII	M	5	2	1	G				

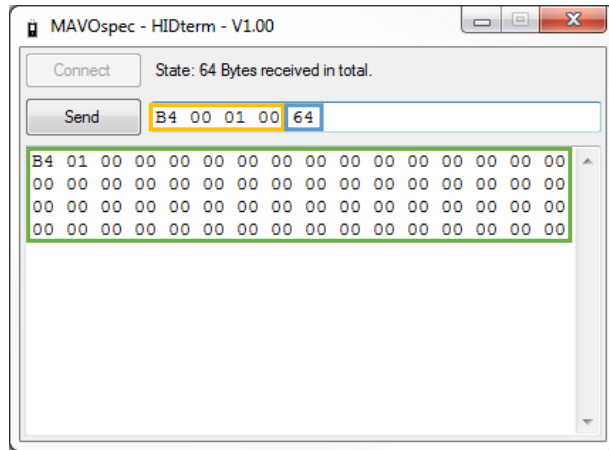
the connection of the conversion HEX -> ASCII is apparent.

The transfer status (second byte in the header) contains information about the correct reception. If the transfer was correct the device will confirm that by sending a HID\_DONE = 0x01. Different replies indicate an error in the transmission. A list with possible transfer statuses is in appendix A.

<sup>2</sup> Remark: Data is displayed in the HEX-Data with LSB first (Little-Endian).

### 3.1.2 Sending Data

If the data to be sent fits within one telegram, the structure is rather simple. It is shown here on the example of setting the display brightness.



You have the **telegram header** consisting of 4 Bytes:

- |                        |                |           |   |
|------------------------|----------------|-----------|---|
| - Command Code         | 1 Byte         | 0xB4      | = CMD_SET_BACKLITE (Set Display Brightness)   |
| - Multi packet command | 1 Byte         | 0x00      | = 0x00 for single packet, 0x01 for multi packet command   |
| - Data index           | 2 Byte (LB HB) | 0x00 0x00 | = Address index of the payload in the target structure (will be ignored for single packet commands) |

The **payload** consists of only one Byte.

- 0x32 = 50%
- 0x64 = 100%

In the **reply** you have the Command Code and Transfer Status followed by zeros to fill the telegram up to 64 Bytes.

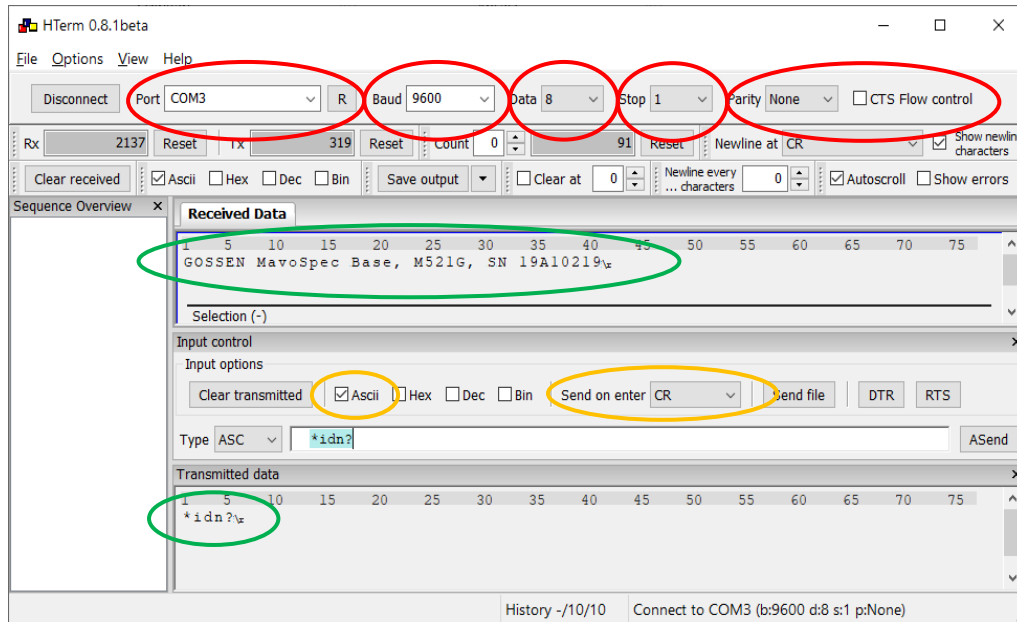
### 3.2 CDC Protocol (Serial Connection)

The UART Settings for the Device are: 9600 Bit per Second, 8 Data bits, 1 Stop bits (no parity, no Flow control)

To check that the serial communication works correctly you can try sending a command to the device with a Terminal-Program. We recommend [HTerm](#) from [der-Hammer.info](#). You can also use a different Terminal-Program that supports sending of several ASCII-Characters in one package.

Make all necessary settings in the Terminal-Program (red Circle). The commands are sent as ASCII and are terminated with either "line feed", "carriage return" or "carriage return - line feed" (yellow circle). The device will reply with the termination characters it received.

If you send „\*idn?“ to the device, the device should answer as shown in the screenshot (green circle).



Additional information:

- CDC commands are not case sensitive
- White spaces have to be placed where necessary
- Some replies are up to 150 Byte long. Make sure your input buffer is big enough and/or check for additional replies after the first one. There is a remark in the command description
- Dates and Times are formatted as set up in the Device Settings
- Data Structures are sent as Text separated by white spaces (very long Structures are broken down in several replies)

**Example 1:**

Command:

sensorinfo?\r

Reply:

The Data from the structure SENSOR\_INFO\_T:

HAMAMATSU HAMAMATSU 256 17K00992 25.02.2019 3.223709e+02 2.419195e+00 -1.621953e-03 2.223414e-06 -2.854980e-08  
5.794171e-11 10.3\r

The parts of the structure are printed with a whitespace between them.

**Example 2:**

Command:

wavelength?\r

Reply:

256 float values:

five values in one reply (to not overfill the buffer), several replies to send all the data:

324.788452 327.202789 329.613922 332.021851 334.426575\r  
336.828094 339.226471 341.621643 344.013702 346.402588\r  
348.788330 351.170929 353.550385 355.926727 358.299957\r  
360.670074 363.037079 365.400940 367.761719 370.119385\r  
....

## 4 Command Descriptions

### 4.1 Read Commands

HID Command Number	CDC Command	Short Command Description
0x01	deviceinfo?	Get Device Information
0x02	sensorinfo?	Get Sensor Information
0x03	fw?	Get Firmware Revision
0x04	hw?	Get Hardware Revision
0x05	Not available (included in Sensor Information)	Get Wavelength Calibration Coefficients
0x06	calibrationdata?	Get Calibration Data complete set
0x07	Not available (included in Calibration Data)	Get Sensor Sensitivity Data
0x08	-	-
0x09	wavelength?	Get Wavelength per Pixel
0x10	-	-
0x11	-	Get Repeat Counter for Averaging used for last measurement Outdated – do not use
0x12	integrationtime?	Get Integration Time in ms used for last measurement
0x13	flags?	Get the Status Flags
0x14	rawspectraldata?	Get Raw spectral data (uncalibrated)
0x15 - 0x19	-	-
0x20	batvoltage?	Get Battery Voltage in mV
0x21	temp?	Get Sensor Temperature
0x22	ambiconversion?	Get single Ambient Sensor Conversion
0x23	doflickermeasure	Start a Flicker Measurement (needed before 0x24 and 0x25)
0x24	ambisensordata?	Get all Ambient Sensor Data (needs 0x23 to be done)
0x25	flickerdata?	Get Calculated Flicker Values (needs 0x23 to be done)
0x26	-	-
0x27	Not available (included in Flicker Values)	Get Flicker Frequency
0x28	calculateddata?	Get Calculated Data
0x29	-	-
0x30	reportitems?	Get currently activated Report Items
0x31	datetime?	Get Device Time and Date
0x32	-	-
0x33	displayfunction?	Get currently Displayed Function
0x34	backlight?	Get current Backlite Setting

#### 4.1.1 Get Device Information

##### HID:

Command to Send:	01 00 00 00
Received Command:	01 01 44 00 [ Data Structure ]
Data Structure:	DEVICE_INFO_T

##### CDC:

Command to Send:	sensorinfo?
Received:	one line, ~65 Byte
Data Structure:	DEVICE_INFO_T

#### 4.1.2 Get Sensor Information

##### HID:

Command to Send:	02 00 00 00
Received Command:	02 01 44 00 [ Data Structure ]
Data Structure:	SENSOR_INFO_T

##### CDC:

Command to Send:	deviceinfo?
Received:	one line, ~130 Byte
Data Structure:	SENSOR_INFO_T

### 4.1.3 Get Firmware Revision

#### HID:

Command to Send:	03 00 00 00
Received Command:	03 01 0A 00 [ Firmware Revision ]
Data Structure:	ASCII-String [10] (V ?.?.?)

#### CDC:

Command to Send:	fw?
Received:	one line, 6 Byte
Data Structure:	Text (x.x.x)

### 4.1.4 Get Hardware Revision

#### HID:

Command to Send:	04 00 00 00
Received Command:	03 01 0A 00 [ Hardware Revision ]
Data Structure:	ASCII-String [10] (15572-??)

#### CDC:

Command to Send:	hw?
Received:	one line, 3 Byte
Data Structure:	Text (xx)

#### 4.1.5 Get Wavelength Calibration Coefficients

##### HID:

Command to Send:	05 00 00 00
Received Command:	05 01 18 00 [ Coefficients ]
Data Structure:	float [6] (A0, B1, B2, B3, B4, B5)

This is part of the Sensor Information.

##### CDC:

Not available in CDC

#### 4.1.6 Get Calibration Data

##### HID:

Command to Send:	06 00 00 00
Received Command:	06 01 28 04 [ Data Structure ]
Data Structure:	<a href="#">DEVICE_CALIBRATION_DATA_T</a>

##### CDC:

Command to Send:	calibrationdata?
Received:	55 lines, <64 Byte each
Data Structure:	<a href="#">DEVICE_CALIBRATION_DATA_T</a>



#### 4.1.7 Get Sensor Sensitivity Data

##### HID:

Command to Send:	07 00 00 00
Received Command:	07 01 04 04 [ Sensitivity Values ]
Data Structure:	float [257]

This is part of the Calibration Data.

##### CDC:

Not available in CDC

#### 4.1.8 Get Wavelength (Pixel)

##### HID:

Command to Send:	09 00 00 00
Received Command:	09 01 04 04 [ Pixel Wavelengths ]
Data Structure:	float [257]

##### CDC:

Command to Send:	wavelength?
Received:	52 lines, <64 Byte each
Data Structure:	float [257] as Text

#### 4.1.9 Get IntegrationTime

##### HID:

Command to Send:	12 00 00 00
Received Command:	12 01 02 00 [ Integration Time ]
Data Structure:	uint16_t (in ms)

##### CDC:

Command to Send:	integrationtime?
Received:	one line, ~7 Byte
Data Structure:	Text (1234) (in ms)

#### 4.1.10 Get the Status Flags

##### HID:

Command to Send:	13 00 00 00
Received Command:	13 01 08 00 [ Data Structure ]
Data Structure:	FLAGS_T

##### CDC:

Command to Send:	flags?
Received:	one line, ~70 Byte
Data Structure:	FLAGS_T as Text, 4 Blocks with 16 Bits separated by one whitespace

#### 4.1.11 Get Raw spectral data

##### HID:

Command to Send:	14 00 00 00
Received Command:	14 01 02 02 [ Uncalibrated Spectral Data ]
Data Structure:	uint16_t [257]

##### CDC:

Command to Send:	rawspectraldata?
Received:	26 lines, <64 Byte each
Data Structure:	uint16_t [257] as Text

#### 4.1.12 Get battery voltage

##### HID:

Command to Send:	20 00 00 00
Received Command:	20 01 02 00 [ Battery Voltage in mV ]
Data Structure:	int16_t

##### CDC:

Command to Send:	batvoltage?
Received:	one line, ~5 Byte
Data Structure:	Volage in mV (1324)

#### 4.1.13 Get sensor temperature

##### HID:

Command to Send:	21 00 00 00	Temperature in °C
	21 00 00 01	Temperature in °F
Received Command:	21 01 04 00 [ Temperature ]	
Data Structure:	float	

##### CDC:

Command to Send:	temp?
Received:	one line, ~7 Byte
Data Structure:	Temperature in °C or °F (as set up in the Device Settings) (13.24 C)

#### 4.1.14 Get single Ambient Sensor Conversion

##### HID:

Command to Send:	22 00 00 00
Received Command:	22 01 02 00 [ Ambient Reading in mV ]
Data Structure:	int16_t

##### CDC:

Command to Send:	ambiconversion?
Received:	one line, <10 Byte
Data Structure:	int16_t as Text

#### 4.1.15 Start a Flicker Measurement (needed before 0x24 and 0x25)

##### HID:

Command to Send:	23 00 00 00
Received Command:	22 01 00 00
Data Structure:	-

##### CDC:

Command to Send:	doflickermeasure
Received:	Echo of the Command
Data Structure:	Text

#### 4.1.16 Get all Ambient Sensor Data (needs 0x23 to be done)

##### HID:

Command to Send:	24 00 00 00
Received Command:	24 01 00 04 [ Values of Flicker Wave ]
Data Structure:	uint16_t [512]

##### CDC:

Command to Send:	ambisensordata?
Received:	52 lines, <64 Byte
Data Structure:	uint16_t [512] as Text

#### 4.1.17 Get calculated flicker values

##### HID:

Command to Send:	25 00 00 00
Received Command:	25 01 18 00 [ Data Structure ]
Data Structure:	FLICKER_T

##### CDC:

Command to Send:	flickerdata?
Received:	one line, <64 Byte
Data Structure:	FLICKER_T

#### 4.1.18 Get Flicker Frequency

##### HID:

Command to Send:	27 00 00 00
Received Command:	27 01 04 00 [ Flicker Frequency in Hz ]
Data Structure:	float

This is part of the Flicker Values.

##### CDC:

Not available in CDC

#### 4.1.19 Get Calculated Data

##### HID:

Command to Send:	28 00 00 00	"All" Calculated Data
	28 00 01 00 01	Spectrum in 5nm Steps
	28 00 01 00 02	CCT, duv, Evis, Ee, LambdaPeak, LambdaPeak_Val, Lambda_Dominant, Purity
	28 00 01 00 03	Color Coordinates (x/y, u/v, u'/v')
	28 00 01 00 04	CRI Values
	28 00 01 00 05	TM30 Values
	28 00 01 00 06	Flicker Data
	28 00 01 00 07	PAR Data
	28 00 01 00 08	TLCI Data
	28 00 01 00 30	Really ALL Calculated Data
Received Command:	28 01 xx xx [ Data Structure ]	
Data Structure:	00: CALCULATED_DATA_T2	
	01: float [4+81]	
	02: float [8]	
	03: CIE_T	
	04: CRI_T	
	05: TM30_T	
	06: FLICKER_T	
	07: PAR_T	
	08: TLCI_DATA_T	
	30: CALCULATED_DATA_T	



## CDC:

Command to Send:	calculateddata? 01	Spectrum in 5nm Steps
	calculateddata? 02	CCT, duv, Evis, Ee, LambdaPeak, LambdaPeak_Val, Lambda_Dominant, Purity
	calculateddata? 03	Color Coordinates (x/y, u/v, u'/v')
	calculateddata? 04	CRI Values
	calculateddata? 05	TM30 Values
	calculateddata? 06	Flicker Data
	calculateddata? 07	PAR Data
	calculateddata? 08	TLCI Data
Received:	01: 18 lines, <64 Byte each	
	02: one line, ~50 Byte	
	03: one line, ~45 Byte	
	04: 3 lines, <64 Byte each	
	05: 23 lines, <64 Byte each	
	06: one line, ~60 Byte	
	07: 12 lines, <64 Byte each	
	08: 16 lines, < 64 Byte each	
Data Structure:	01: float [81] as Text	
	02: float [8] as Text	
	03: CIE_T	
	04: CRI_T	
	05: TM30_T	
	06: FLICKER_T	
	07: PAR_T	
	08: TLCI_DATA_T	

#### 4.1.20 Get currently activated Report Items

##### HID:

Command to Send:	30 00 00 00
Received Command:	30 01 04 00 [ Data Structure ]
Data Structure:	REPORTAUSWAHL

##### CDC:

Command to Send:	reportitems?
Received:	one line, 32 Byte
Data Structure:	REPORTAUSWAHL, 32 single Bits as Text

#### 4.1.21 Get Device Time and Date

##### HID:

Command to Send:	31 00 00 00
Received Command:	30 01 07 00 [ Data Structure ]
Data Structure:	BCDTIME_T

##### CDC:

Command to Send:	datetime?
Received:	one line, <30 Byte
Data Structure:	Text: "Time:---- Date:----" each in the format set up in the Device Settings

#### 4.1.22 Get currently Displayed Function

##### HID:

Command to Send:	33 00 00 00
Received Command:	33 01 01 00 [ Data Structure ]
Data Structure:	PROGSTATES

##### CDC:

Command to Send:	displayfunction?
Received:	one line, ~1 Byte
Data Structure:	PROGSTATES

#### 4.1.23 Get current Backlite Setting

##### HID:

Command to Send:	34 00 00 00
Received Command:	33 01 01 00 [ Brightness Setting ]
Data Structure:	uint8_t (Brightness in %)

##### CDC:

Command to Send:	backlight?
Received:	one line, ~3 Byte
Data Structure:	uint8_t (Brightness in %)

## 4.2 Write Commands

HID Command Number	CDC Command	Short Command Description
0xB0	reportitems	Set activate Report Items <sup>3,4</sup>
0xB1	datetime	Set Device Time and Date
0xB2	keycode	Send Keystroke
0xB3	displayfunction	Set Function to be displayed
0xB4	backlight	Set Display Brightness <sup>3,4</sup>
0xB5	keylock	Set Keylock <sup>3</sup>
0xB6 - 0xC0	-	-
0xC1	savesettings	Save Settings in EEPROM
-	beep	Let the device make a beep

---

<sup>3</sup> Remark: You can temporarily overwrite data and settings with these functions (not saved - resets after device restart)

<sup>4</sup> Remark: These functions can be permanently saved by using command 0xC1 / savesettings

#### 4.2.1 Set activate Report Items

##### HID:

Command to Send:	B0 00 00 00 [ Data Structure ]
Received Command:	B0 01 00 00 ...
Data Structure:	REPORTAUSWAHL

##### CDC:

Command to Send:	reportitems [ Data Structure ]
Received:	Echo of the Command
Data Structure:	REPORTAUSWAHL

#### 4.2.2 Set Device Time and Date

##### HID:

Command to Send:	B1 00 00 00 [ Data Structure ]
Received Command:	B1 01 00 00 ...
Data Structure:	BCDTIME_T

##### CDC:

Command to Send:	datetime [ Data Structure ]
Received:	Echo of the Command
Data Structure:	"hh:mm dd.mm.yyyy" fixed in this format

### 4.2.3 Send Keystroke

#### HID:

Command to Send:	B2 00 00 00 [ Data Structure ]
Received Command:	B2 01 00 00 ...
Data Structure:	KEYCODES

#### CDC:

Command to Send:	keycode [ Data Structure ]
Received:	Echo of the Command (including the keycode)
Data Structure:	KEYCODES

### 4.2.4 Set Function to be displayed

#### HID:

Command to Send:	B3 00 00 00 [ Data Structure ] [Sub-Window-Selection]
Received Command:	B3 01 00 00 ...
Data Structure:	PROGSTATES      Sub-Window-Selection: 0, 1, 2, ...

"Sub-Window-Selection" is e.g. which CRI you want to show (Bar, Table, Web) or CIE standard or zoomed.

#### CDC:

Command to Send:	backlight [ Data Structure ]
Received:	Echo of the Command
Data Structure:	PROGSTATES      Sub-Window-Selection: 0, 1, 2, ...

#### 4.2.5 Set Display Brightness

##### HID:

Command to Send:	B4 00 00 00 [ Brightness ]
Received Command:	B4 01 00 00 ...
Data Structure:	uint8_t (Brightness in %)

##### CDC:

Command to Send:	displayfunction [ Brightness ]
Received:	Echo of the Command
Data Structure:	Text (Brightness in %, without percent sign)

#### 4.2.6 Set Keylock

##### HID:

Command to Send:	B5 00 00 00 [ Keylock enable ]
Received Command:	B5 01 00 00 ...
Data Structure:	uint8_t (0 = no Keylock)

This is not permanent and will be reset after a device restart.

##### CDC:

Command to Send:	keylock [ Keylock enable ]
Received:	Echo of the Command
Data Structure:	No Keylock: "0" or "false"      Keylock: "1" or "true"

#### 4.2.7 Save Settings in EEPROM

##### HID:

Command to Send:	C1 00 00 00 00
Received Command:	C1 01 00 00 ...
Data Structure:	none (everything else than 0x00 will result in an error)

##### CDC:

Command to Send:	savesettings
Received:	Echo of the Command
Data Structure:	none

#### 4.2.8 Beep

##### HID:

Not available

##### CDC:

Command to Send:	beep [ Data Structure ]
Received:	Echo of the Command
Data Structure:	1: short beep 2: long beep 3: error beep Anything else = short beep



### 4.3 Example: How to start a Measurement and get the Data

To get measured data you basically have to start a measurement, wait for the device to finish measuring and calculating and then request the calculated data.

Before sending the keystroke you could check that the device is ready and responsive by requesting the status flags but you don't have to. This would only be to be double sure.

#### 1. Send keystroke for measurement key

Send "B2 00 01 00 40" to the device (see paragraph 4.2.7).

By simulating the keystroke for the "M-Key" on the device, you start a measurement the way you would start it by hand.

If the device is ready to take a measurement you immediately get the response of the device, that the request was accepted.

#### 2. Wait till the device finished calculating

Send "13 00 00 00" to get the status flags from the device (see paragraph 4.1.11).

You can send that command while the device is taking the measurement and is calculating. The Command is stored in the buffer and the device will answer when it has finished.

You could basically send any command and wait for a response of the device. We recommend however to check the status flags to see if the measurement and calculation resulted in the desired outcome as it is possible that the calculation failed e.g. because the signal was too strong.

#### 3. Get calculated data

Send "28 00 00 00" (or any other 28... command) to request the calculated data you want (see paragraph 4.1.20).

You "just" have to fill the structure with the received data to easily access the desired data.

## Appendix A Possible Transfer Statuses and Error Codes

### HID:

Transfer Status	Status Name	Status Description
0x01	HID_DONE	Successful Transmission
0x02	HID_PENDING_DATA	Not used
0x03	HID_ERROR_TIMEOUT	Not used
0x04	HID_ERROR_INVALID_CMD	The Sent Command was not recognized
0x05	HID_ERROR_INVALID_LENGTH	The specified data length does not match the expected length
0x06	HID_ERROR_INVALID_ADDRESS	Not used
0x07	HID_ERROR_INVALID_PARAM_VALUE	The specified parameter value is not supported
0x08	HID_ERROR_INVALID_TERMINATION	Not used
0x09	HID_ERROR_WRONG_PASSWORD	Some functions are password protected (reserved for production)

### CDC:

Error Code	Status Description
1	Command not Recognized
2	Parameter not supported

## Appendix B      Data Structures

BCDTIME\_T, Structure for Time / Date, 1 Byte each as BCD

```
typedef struct {
    uint8_t      tm_sec;
    uint8_t      tm_min;
    uint8_t      tm_hour;
    uint8_t      tm_weekday;
    uint8_t      tm_day;
    uint8_t      tm_mon;
    uint8_t      tm_year;
} BCDTIME_T;
```

PROD\_STATE\_T, Service- and Production States

```
typedef enum {
    DEV_LOCKED,                // FBGS come from production, not tested
    DEV_FACTORY_TEST_OK,      // Function tests passed, sensor data saved
    DEV_CALIBRATION_OK        // Device is fully calibrated and operational}
PROD_STATE_T;
```

KEYCODES, KeyCodes for CMD\_SET\_KEYCODE

```
typedef enum {
    KEY_UP      = 0x01,    up
    KEY_DOWN    = 0x02,    down
    KEY_LEFT    = 0x04,    left
    KEY_RIGHT   = 0x08,    right
    KEY_MENU    = 0x10;    menu
    KEY_MEM     = 0x20,    mem
    KEY_M       = 0x40     m
} KEYCODES;
```

#### DEVICE\_INFO\_T, Structure of the Device Information

```
typedef struct {  
    char            DeviceID[10];           // M511G  
    char            Manufacturer[16];       // GOSSEN GmbH  
    char            DeviceDescriptor[16];   // MAVOspec BASE  
    char            SerialNumber[10];       // 14B03386  
    BCDTIME_T       ProductionDate;         // 16/05/29  
    BCDTIME_T       LastServiceDate;        // 18/02/10  
    PROD_STATE_T    ProdState;              // 0..x  
} DEVICE_INFO_T;
```

#### SENSOR\_INFO\_T, Structure for Sensor Information

```
typedef struct {  
    SENSOR_SPECS_T  SensorType;  
    char            SerialNumber[10];       // 12x000123  
    BCDTIME_T       InspectionDate;         // 13/12/29  
    float           WaveCalCoeff[6];        // A0,B1,B2,B3,B4,B5 Attn !! XC16 float only !!  
    float           FWHM;                   // xx.y  
} SENSOR_INFO_T;
```

#### SENSOR\_SPECS\_T, Structure for Sensor Information

```
typedef struct {  
    char            VendorDescriptor[10];   // Hamamatsu  
    char            SensorDescriptor[10];   // C12666MA  
    uint16_t        NumPixels;              // 256  
} SENSOR_SPECS_T;
```

## REPORTAUSWAHL, Structure of the Items to show in the Report Window

```
Union {
    uint32_t    state;
    struct {
        unsigned Evis:1;
        unsigned Eenerg:1;
        unsigned LER:1;
        unsigned CCT:1;
        unsigned duv:1;
        unsigned x:1;
        unsigned y:1;
        unsigned u:1;
        unsigned v:1;
        unsigned u_:1;
        unsigned v_:1;
        unsigned CRIRa:1;
        unsigned CRIRe:1;
        unsigned CRIGAl:1;
        unsigned TM30Rf:1;
        unsigned TM30Rg:1;
        unsigned lpeak:1;
        unsigned ldomi:1;
        unsigned purity:1;
        unsigned flicker_i:1;
        unsigned flicker_p:1;
        unsigned flicker_f:1;
        unsigned PAR:1;
        unsigned TLCI:1;
        unsigned TLMF:1;
        unsigned bit_26:1 bit 26 to bit 32 are not in use but 32bit will always be sent or expected
        unsigned bit_27:1 as the uint32_t state of the union is used for sending and receiving
        unsigned bit_28:1
        unsigned bit_29:1
        unsigned bit_30:1
        unsigned bit_31:1
        unsigned bit_32:1
    };
} REPORTAUSWAHL;
```

CALCULATED\_DATA\_T2, Structure of the Measured Data (for CMD\_GET\_CALCULATED\_DATA with Parameter 00)

```
typedef struct {  
    bool            Geladen;  
    float           Spectrum[85];           // Spectrum (380nm-780nm) in 5nm Steps  
    float           Ra;  
    float           Rx[15];  
    float           CIE_x;  
    float           CIE_y;  
    float           CIE_u;  
    float           CIE_v;  
    float           CIE_u_;  
    float           CIE_v_;  
    float           CCT;  
    float           duv;  
    float           Lux;  
  
    float           Flicker_E_Min;  
    float           Flicker_E_Max;  
    float           Flicker_E_Avr;  
    float           Flicker_Index;  
    float           Flicker_Percent;  
    float           Flicker_Frequency;  
  
    uint16_t        LambdaP;                // Peak Wavelength  
    float           LambdaP_val;            // Value of Peak Wavelength  
    int16_t          domLambda;             // Dominant Wavelength  
    float           purity;  
} CALCULATED_DATA_T2;
```

# CALCULATED\_DATA\_T, Structure of the Measured Data (for CMD\_GET\_CALCULATED\_DATA with Parameter 30)

```
typedef struct {
    float      Spectrum[85];           // Spectrum (380nm-780nm) in 5nm Steps

    TM30_T      tm30Data;

    CRI_T       CRI;

    CIE_T       CIE;

    FLICKER_T    Flicker;

    TLCL_DATA_T  TLCL;

    FLAGS_T     Flags;

    float      CCT;
    float      duv;
    float      Evis;
    float      Ee;
    float      LER;

    uint16_t    LambdaP;               // Peak Wavelength
    float      LambdaP_val;            // Value of Peak Wavelength
    int16_t     domLambda;              // Dominant Wavelength
    float      purity;
    uint16_t    Integrationszeit;       // Integrationtime
    uint16_t    Mittelwertanzahl;       // Averaging

    PAR_T       Pflanzenwerte          // PAR
} CALCULATED_DATA_T;
```

[Subtypes on the following pages](#)

#### HUEBIN\_T, Subtype for [CALCULATED\\_DATA\\_T](#)

```
typedef struct {  
    int16_t      m;           // number of CES which fall into this huebin  
    float        aM;          // averaged a' of all CES in this huebin  
    float        bM;          // averaged b' of all CES in this huebin  
} HUEBIN_T;
```

#### TM30\_T, Subtype for [CALCULATED\\_DATA\\_T](#)

```
typedef struct {  
    float        Rf;  
    float        Rg;  
    HUEBIN\_T    HueBinT[16];    // Test  
    HUEBIN\_T    HueBinR[16];    // Reference  
    int8_t       Chromashift[16]; // Chromashift in %  
    int8_t       Hueshift[16];   // Hueshift in %  
    uint8_t      Fidelity[16];   // FidelityIndex by Hue (0..100)  
} TM30_T;
```

#### CRI\_T, Subtype for [CALCULATED\\_DATA\\_T](#)

```
typedef struct {  
    float        Ra;  
    float        Re;  
    float        GAI;  
    float        Rx[15];  
} CRI_T;
```

#### CIE\_T, Subtype for [CALCULATED\\_DATA\\_T](#)

```
typedef struct {  
    float        x;  
    float        y;  
    float        u;  
    float        v;  
    float        u_;  
    float        v_;  
} CIE_T;
```



## FLAGS\_T, Subtype for CALCULATED\_DATA\_T

```
typedef struct {
    union {
        uin16_t      All;
        struct {
            unsigned   AutoExposureOver:1;
            unsigned   AutoExposureUnder:1;
            unsigned   SensorDynamicsOver:1;
            unsigned   SensorDynamicsUnder:1;
        };
    } measure;
    union {
        uin16_t      All;
        struct {
            unsigned   AllCalc:1;           // No calculation performed
            unsigned   Spectrumfail:1;      // Measurement failed
            unsigned   CCTfail:1;           // CCT invalid, because:
            unsigned   CCTover:1;           // CCT > 50.000 K
            unsigned   CCTunder:1;          // CCT < 1600 K
            unsigned   CCTduv:1;            // duv < -0,1
            unsigned   CRIfail:1;           // CIE invalid
            unsigned   TM30fail:1;          // TM30-Values invalid
            unsigned   Flickerfail:1;       // Flicker invalid
        };
    } calc;
    union {
        uint16_t     All;
        struct {
            unsigned   freq_zu_klein:1;     // Frequency to low
            unsigned   freq_zu_gross:1;      // Frequency to high
            unsigned   wert_zu_klein:1;      // Flicker intensity to low
            unsigned   wert_zu_gross:1;      // Flicker intensity to high
        };
    } flicker;
    union {
        uint16_t     All;
        struct {
            unsigned   geladen:1;            // Measurement was loaded
            unsigned   messen:1;             // Device is currently taking a measurement
            unsigned   rechnen:1;           // Device is currently calculating
            unsigned   fertig:1;            // Device has finished calculating and is ready
        };
    };
} status;
} FLAGS_T;
```

#### FLICKER\_T, Subtype for [CALCULATED\\_DATA\\_T](#)

```
typedef struct {  
    float      E_Min;  
    float      E_Max;  
    float      E_Avr;  
    float      Index;  
    float      Percent;  
    float      Frequency;  
} FLICKER_T;
```

#### PAR\_T, Subtype for [CALCULATED\\_DATA\\_T](#)

```
typedef struct {  
    float      PPFD;  
    float      PPFD_UV;           // UV  
    float      PPFD_B;           // Blue  
    float      PPFD_G;           // Green  
    float      PPFD_R;           // Red  
    float      PPFD_FR;          // Far Red  
} PAR_T;
```

#### TLCI\_SECTOR\_T, Subtype for [CALCULATED\\_DATA\\_T](#)

```
typedef struct {  
    int8_t lightness;  
    int8_t chroma;  
    int8_t hue;  
} TLCI_SECTOR_T;
```

TLCI\_DATA\_T, Subtype for [CALCULATED\\_DATA\\_T](#)

```
typedef struct{
    int Qa1;
    int Qa2;
    int Qa3;
    int Qa4;
    int Qa5;
    int Qa6;
    int Qa7;
    int Qa8;
    int Qa9;
    int Qa10;
    int Qa11;
    int Qa12;
    int Qa13;
    int Qa14;
    int Qa15;
    int Qa16;
    int Qa17;
    int Qa18;
    double Duv;
    char ReferenceType;
    uint8_t TLCI
    TLCI\_SECTOR\_T R;
    TLCI\_SECTOR\_T RY;
    TLCI\_SECTOR\_T Y;
    TLCI\_SECTOR\_T YG;
    TLCI\_SECTOR\_T G;
    TLCI\_SECTOR\_T GC;
    TLCI\_SECTOR\_T C;
    TLCI\_SECTOR\_T CB;
    TLCI\_SECTOR\_T B;
    TLCI\_SECTOR\_T BM;
    TLCI\_SECTOR\_T M;
    TLCI\_SECTOR\_T MR;
}TLCI_DATA_T;
```

#### DEVICE\_CALIBRATION\_DATA\_T, Structure of Calibration Data

```
typedef struct {
    float          Sensitivity[257];          // Sensitivity of every pixel
    float          DarkSignalTCoeff_1;        // BETA: linear coefficient
    float          DarkSignalTCoeff_2;        // GAMMA: exp coefficient

    uint16_t       SensorSaturationValue;     // Max Pixel Value, OverloadDetection
    int16_t        SensorOffset;              // Min Pixel Value, without darksignal f(Tcal,tint=0)
    float          SensorTCoeff;              // Off-TemperatureCoeff, Slope = f(Tact)
    float          Tcal;                     // SensorTemperature at calibration time
    int16_t        AmbiDarkValue;
    int16_t        AmbiSensitivity;

    float          PhotConvFactor;            // Radiometric [ W/m² ]-> km -> Photometric [Ix]
    BCDTIME_T      SensCalDate;              // 16/06/29
} DEVICE_CALIBRATION_DATA_T;
```

#### PROGSTATES, Program States

```
typedef enum {
    PM_SETTINGS,          // 1
    PM_MEMORY,            // 3

    PM_SPECTRUM,          // 6
    PM_REPORT,            // 7
    PM_REPORT_REF,        // 8 (Reference Mode)
    PM_CRI,               // 9
    PM_TM30,              // 10
    PM_FLICKER,           // 11
    PM_DATA,              // 12
    PM_CIE,               // 13
} PROGSTATES;
```

## Document Revision History

Version	Date	Creator	Short Description
V 0.11	25.07.2016	Bg	First Revision
V 1.0	16.02.2017	Bg	Update for MAVOSPEC <b>BASE</b> Firmware V 1.1.x
V 2.0	22.05.2018	HK	Design change and Update for MAVOSPEC <b>BASE</b> (Firmware V 1.2.3 and newer)
V 2.1	19.11.2018	HK	Update for Firmware V 1.2.3 Release
V 3.0	29.01.2021	HK	Update for Serial Connection and added TLCl information (Firmware V1.3.0 Release)