Muse v3

Communication Protocol

Revision History

Date	Revision	Author	Description
20/01/2022	1.0	DC	First draft
26/01/2022	1.1	DC	Update with first revision of BLE streaming mode
02/02/2022	1.2	DC	Battery voltage command
10/02/2022	1.3	MP	Updated log frequency command & changed data mode size
17/02/2022	1.4	MP	Updated memory commands
18/02/2022	1.5	MP	Updated information on log and sensors full-scale
21/02/2022	1.6	MP	Added information on data offload
21/02/2022	1.7	DC / MP	Updated data table
23/02/2022	1.8	MP	Updated ack of STATE command
18/03/2022	1.9	MP	Separated gyro and axl modes
28/03/2022	2.0	MP	Added calibration procedure and inverted GYRO and AXL codes
29/04/2022	2.1	MP	Added missing data formats (environmental sensors)
04/05/2022	2.2	MP	Added magnetometer data decode for hw rev 2, fixed error on quaternions decode
18/05/2022	2.3	MP	Added standby and circular memory config, conditional log
27/05/2022	2.4	MP	Moved all user config to single command (0x51)
08/06/2022	2.5	MP	Fixed streaming command response
07/07/2022	2.6	MP	Conditional log information
14/12/2022	2.7	RB	Major revision
09/01/2023	2.8	MP	Microphone information
11/01/2023	2.9	RB	Overall command review, examples, format minor revision
16/01/2023	2.10	RB	File Info command, Sensor full scale management review, minor fix
17/01/2023	2.11	RB	Data acquisition source code examples, calib procedure and minor fix
27/01/2023	2.12	RB	Minor fix to sensor full scales management
28/01/2023	2.13	RB	Minor fix, BLE version decoding, API Java porting
09/03/2023	2.14	RB	Updated user settings management
17/03/2023	2.15	RB	Added reset functionality to restart command for factory reset management

Table of Contents

1 Introduction

This document describes the communication protocol designed in the framework of the Muse v3 project. This protocol can be used with the wireless (Bluetooth Low Energy, BLE) interface to retrieve parameters from the system or to apply changes to the system, as well as to control the data acquisition and device calibration.

To control the device using the wired interface (i.e., USB), each message must be wrapped with a unique header and trailer, as described in Section 2.2.

2 Communication protocol

2.1 Overview

The system designed in the framework of this project complies with a bidirectional communication protocol based on the Type-Length-Value (TLV) format. As summarized in Table 1, each packet contains one byte indicating the type (e.g., command), one byte indicating the number of bytes in the value (e.g., the length of the payload), and the value (e.g., the payload itself).

Type	**Length**	**Value (payload)**		
8-bit unsigned int	8-bit unsigned int	Byte-0	....	Byte-N

TABLE 1: TLV FORMAT.

Each TLV packet shall be acknowledge by the receiver with an ACK response, either positive (success) or negative (error), carrying information about the outcome using the same TLV format. As summarized in Table 2, the ACK response contains the ACK_CODE which identifies that the message is actually an ACK, the length of the value, and the value. The length of the value is at least equal to 2, for it carries information about which message the ACK refers to and the error code. In case the command sent previously needs some information back, the bytes following the error code byte contain the actual information.

Type	**Length**	**Value (payload)**		
ACKNOLEDGE CODE	8-bit unsigned int	COMMAND CODE	ERROR CODE	…

TABLE 2: STRUCTURE OF THE ACKNOWLEDGE MESSAGE.

Note: the **ACKNOWLEDGE COMMAND CODE** corresponds to 0x00.

The ERROR CODE can be:

- 0x00 ACK OK Positive acknowledge.
- 0x01 ACK KO Negative acknowledge.

*The **ERROR CODE** field is common to all acknowledge messages. See Section 5 (Commands) for a detailed description of each command structure.*

Each command byte has a SET/GET bit, which can be used either to change the parameter / behavior related to the command or to retrieve it, respectively. The bit used for such purpose is the Most Significant bit (MSb), as summarized in Table 3.

Command type			
b7 – R/W Read (R) = 1 Write (W) = 0	b6	....	b0

TABLE 3: COMMAND TYPE ARRANGEMENT.

In practice this is obtained by adding the value **0x80** to the hex code of the command to be executed in read (GET) mode.

The Muse **Generic ATTribute Profile (GATT)** defines how data are transferred back and forth with respect to an external device (i.e., typically mobile or desktop applications), using Services and Characteristics. Standard services and characteristics of the BLE profile are highlighted in light grey. A standard **Device Information Service** provides access to generic manufacturing and system information. Finally, a custom service, named **Muse_v3 Service**, allows to control and access data. By means of the two characteristics provided by the latter service, **Command** and **Data**, it is possible to send and receive a command to the system and to retrieve data, respectively. All information is exchanged by means of a bidirectional protocol.

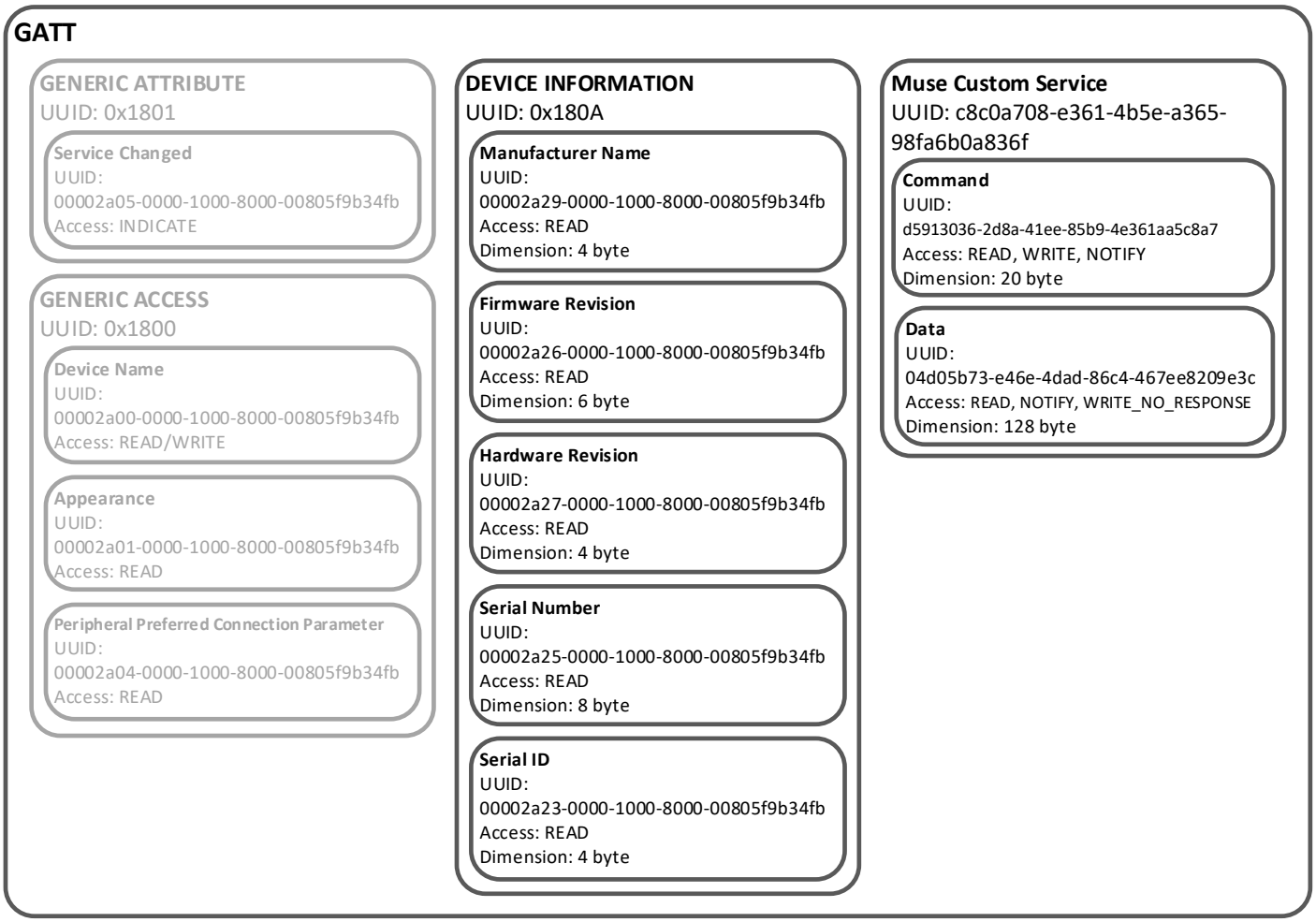

FIGURE 1: GATT PROFILE OF MUSE V3.

Numeric values are managed using LITTLE-ENDIAN byte order.

Byte fields of buffer and messages are in HEX format.

2.2 USB: Header and Trailer

All commands and messages described in this document are available through both wireless (i.e., Bluetooth Low Energy) and wired (i.e., serial / USB) interfaces. It is sufficient to wrap the default messages, provided in this document for BLE interface, with a custom header (“?!”) and trailer (“!?”) ASCII patterns to be used with wired interface.

	Header		Message Body	Trailer	
Index	0	1	*Byte array variable length*	N-1	N
ASCII	‘?’	‘!’	*Byte array representing the command to be sent to the device or the message received from the device itself*	‘!’	‘?’
HEX	0x3f	0x21		0x21	0x3f

3 Device Information Service

The Device Information Service exposes the manufacturer information. It is declared as a Primary Service. It is not dependent upon any other services. The service **UUID** is **0x180A**. The Device Information Service exposes 5 characteristics: Manufacturer Name, Firmware Revision, Hardware Revision, Serial Number, System ID.

3.1 Manufacturer Name

UUID: 00002a29-0000-1000-8000-00805f9b34fb
Byte: 4
Property: READ
Value: 221e
Description: UTF-8 string representing the name of the manufacturer of the device.

3.2 Firmware Revision

UUID: 00002a26-0000-1000-8000-00805f9b34fb
Byte: 6
Property: READ
Value: Firmware version described with the format *[Major].[Minor].[Patch]*. Example: 1.5.06.
Description: UTF-8 string representing the firmware revision currently installed on device.

3.3 Hardware Revision

UUID: 00002a27-0000-1000-8000-00805f9b34fb
Byte: 4
Property: READ
Value: 3.0
Description: UTF-8 string representing the hardware revision of device.

3.4 Serial Number

UUID: 00002a25-0000-1000-8000-00805f9b34fb
Byte: 8
Property: READ
Value: *[8-bytes alphanumeric code]*. Example: 0346b583.
Description: UTF-8 string representing a unique system identifier. It allows us to uniquely identify each board.

3.5 System ID

UUID: 00002a23-0000-1000-8000-00805f9b34fb
Byte: 4
Property: READ
Value: 03b1
Description: UTF-8 string representing the system identifier using the device model, as encoded by the manufacturer.

*All the above fields are set during the end-of-line testing and programming routines execution. They are not changed for the whole life of each device. Only **Firmware Revision** is suitably updated every device firmware update.*

***Serial Number** and **System ID** allows us to uniquely identify each board guaranteeing complete traceability from assembly and testing to storage and sale of each device.*

*Revision fields are coded according to a three-part version number: **Major.Minor.Patch**. Breaking changes are indicated by increasing the major number (e.g., new features that significantly impact on device behavior). New, non-breaking features increase the minor number (e.g., new features that don't significantly change the behavior of the device such as a new command response or introduction of a new metrics). All other non-breaking changes increment the patch number (e.g., typically bug fixes).*

4 Muse Custom Service

The Muse Custom Service exposes the commands and functionalities necessary to control the device and access data. It is declared as a Primary Service. It is not dependent upon any other services. The service **UUID** is **c8c0a708-e361-4b5e-a365-98fa6b0a836f**. The Muse Custom Service exposes 2 characteristics: Command and Data.

4.1 Command Characteristic

UUID:	d5913036-2d8a-41ee-85b9-4e361aa5c8a7
Byte:	20
Property:	READ, WRITE, NOTIFY
Description:	It allows to control the device behaviors.

4.2 Data Characteristic

UUID:	09bf2c52-d1d9-c0b7-4145-475964544307
Byte:	128
Property:	READ, NOTIFY, WRITE_NO_RESPONSE
Description:	It allows to manage streaming of data from device.

5 Commands

Table 4 provides a list of commands supported by the Muse v3 device.

Name	R/W	Code (hex)	Code (dec)	Description
CMD_ACK	W	0x00	0	Acknowledge
CMD_STATE	W	0x02	2	Set system state
	R	0x82	130	Get system state
CMD_RESTART	W	0x03	3	Restart device
CMD_APP_INFO	R	0x84	132	Get application firmware information (i.e., CRC and length)
CMD_BATTERY_CHARGE	R	0x87	135	Get battery charge level
CMD_BATTERY_VOLTAGE	R	0x88	136	Get battery voltage
CMD_CHECK_UP	R	0x89	137	Read value of check-up register
CMD_FW_VERSION	R	0x8A	138	Get bootloader and application firmware versions
CMD_TIME	W	0x0B	11	Set date / time
	R	0x8B	139	Get date / time
CMD_BLE_NAME	W	0x0C	12	Set BLE device name
	R	0x8C	140	Get BLE device name
CMD_DEVICE_ID	R	0x8E	142	Get device unique identifier
CMD_DEVICE_SKILLS	R	0x8F	143	Get device skills in terms of hardware and software features
CMD_MEM_CONTROL	W	0x20	32	Erase memory
	R	0xA0	160	Get memory status
CMD_MEM_FILE_INFO	R	0xA1	161	Get file information
CMD_MEM_FILE_DOWNLOAD	W	0x22	34	Manage file download
CMD_CLK_OFFSET	W	0x31	49	Set a new clock offset
	R	0xB1	177	Retrieve current clock offset
CMD_TIME_SYNC	W	0x32	50	Enter time sync procedure
	R	0xB2	178	Update time sync variables
CMD_EXIT_TIME_SYNC	W	0x33	51	Exit time sync procedure
CMD_SENSORS_FS	W	0x40	64	Set device full scales
	R	0xC0	192	Get device full scales
CMD_MATRIX_CALIBRATION	W	0x48	72	Set calibration matrix
	R	0xC8	200	Get calibration matrix
CMD_BTN_LOG	W	0x50	80	Set button log configuration
	R	0xD0	208	Get button log configuration
CMD_USER_CFG	W	0x51	81	Set user configuration
	R	0xD1	209	Get user configuration

TABLE 4: COMMAND SUPPORTED BY MUSE V3 DEVICE.

In the examples and tables shown from here on, for simplicity, the "0x" prefix will be removed from the hexadecimal (HEX) codes.

5.1 General

5.1.1 Restart Device

The restart command can be executed in write mode only. The HEX code is: 0x03. To perform a restart operation, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	03	01	RESTART MODE	0 … 0																

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	02	03	ERROR CODE	0 … 0															

- RESTART MODE
  It must be one of the following:
  - 0x00 APPLICATION Restart device in application mode
  - 0x01 BOOT Restart device in boot loader mode
  - 0x02 RESET Restart device in application mode after performing a factory reset

Performing a restart operation (i.e., message: 0x03 0x01 0x02) using RESTART MODE equal to 0x02 (RESET) the device configurations are reset to the factory default settings. Calibration matrices and memory contents are permanently erased. When used with MPE functionality, the system MUST be calibrated again.

5.1.2 Application Firmware Information

The application information command can be executed in read mode only. The HEX code is: 0x84. To access the application information, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	84	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	0A	84	ERROR CODE	APP CRC				APP LENGTH				0 … 0							

- APP CRC
  It represents the Circular Redundancy Check (CRC) as 32-bit unsigned integer.

- APP LENGTH
  It represents the length of the application firmware (i.e., number of bytes), as a 32-bit unsigned integer.

Example:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
					APP CRC				APP LENGTH											
Value	00	0A	84	00	53	E9	63	CA	48	90	02	00	0 … 0							

- APP CRC: “53 E9 63 CA” which corresponds to 3395545427.
- APP LENGTH: “48 90 02 00” which corresponds to 168008 bytes.

5.1.3 Device Check-up

The device check-up command can be executed in read mode only. The HEX code is: 0x89. To get the current check-up register value, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	89	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	04	89	ERROR CODE	CHECK-UP REGISTER	0 … 0														

- CHECK-UP REGISTER
  It represents the health status of the device in relation to its peripheral configuration. The 16-bit unsigned integer must be decoded performing a bitwise operation using the following mask:

Byte Order	0 (index 4 in RESPONSE)								1 (index 5 in RESPONSE)							
Bit Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	LSE	HSE	BTN	IMU	MAG	HDR	MEM	BAR	PRX	HUM	BAT	MIC	RFU			

TABLE 5: BIT-MASK FOR CHECK-UP REGISTER DECODING.

Each bit-field can be 0 (normal condition) or 1 (error condition).

5.1.4 Firmware Version

The firmware version command can be executed in read mode only. The HEX code is: 0x8a. To access the device firmware version, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	8A	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	12	8A	ERROR CODE	BOOT LOADER FW VERSION				APPLICATION FW VERSION										BT VERSION MAJOR	BT VERSION MINOR

- BOOT LOADER FW VERSION and APPLICATION FW VERSION
  They are provided in HEX format and must be converted into the corresponding ASCII Text String. They come with the termination "\0" character and represent the boot loader and application firmware versions currently installed on device. If no application firmware is installed, the APPLICATION FW VERSION field is "0.0.00".
- BT VERSION MAJOR and BT VERSION MINOR
  They are provided in HEX format and correspond to the major and minor version numbers of the flashed BLE stack.

Example:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
					BOOT LOADER FW VERSION							APPLICATION FW VERSION							BLE MAJ	BLE MIN
Value	00	10	8A	00	31	2E	33	2E	30	31	00	31	2E	35	2E	32	32	00	01	0B

- BOOT LOADER FW VERSION: "31 2E 33 2E 30 31" which corresponds to "1.3.01".
- APPLICATION FW VERSION: "31 2E 35 2E 32 32" which corresponds to "1.5.22".
- BLE STACK VERSION "01 0B" which corresponds to "1.11"

The end-of-string character "\0" has been removed from both values before conversion to ASCII.

5.1.5 Date/Time

The date/time command can be executed in both write and read mode. The HEX code is 0x0B (write), and 0x8B (read).

To **SET** a new date/time, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	0B	04	TIMESTAMP TO BE SET				0 … 0													

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	02	0B	ERROR CODE	0 … 0															

- TIMESTAMP TO BE SET
  It is the 32-bit unsinged integer value that represents the timestamp to be set, in Unix epoch format.

To **GET** the current date/time, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	8B	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	06	8B	ERROR CODE	TIMESTAMP				0 … 0											

- TIMESTAMP
  It is the 32-bit unsigned integer value that represents the current date/time as retrieved from the device, in Unix epoch format.

Example:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
					TIMESTAMP				0 … 0											
Value	00	06	8B	00	00	FA	BF	63												

- TIMESTAMP: “00 FA BF 63” which corresponds to “1/12/2023 12:16:00 PM”.

The date / time is provided in **UTC (GMT) universal time format**, with **seconds resolution**.

5.2 Device

A set of dedicated commands allow to access the Bluetooth Low Energy (BLE) device name, the unique identifier (ID) alphanumeric code and the hardware and software features exposed by the system itself.

5.2.1 Device Name

The BLE name command can be executed in both write and read mode. The HEX code is 0x0C (write), and 0x8C (read).

To **SET** a custom name, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	0C	PAYLOAD LENGTH	DEVICE NAME TO BE SET																0 … 0	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	02	0C	ERROR CODE	0 … 0															

- PAYLOAD LENGTH
  The overall payload length, as 8-bit unsigned integer. It corresponds to the length (i.e., number of characters, including the end-of-string character) of the device name to be set.

- DEVICE NAME TO BE SET
  The HEX representation of the device name to be set, converted from the corresponding ASCII Text String. It must also include the end-of-string character "\0".

To **GET** the current device name, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	8C	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	PAYLOAD LENGTH	8C	ERROR CODE	DEVICE NAME															

- PAYLOAD LENGTH
  It is equal to the length of the current device name plus 2 (i.e., due to the command and error code fields, in the response buffer).

- DEVICE NAME
  The HEX representation of the current device name. It must be converted into the corresponding ASCII Text String. It also includes the end-of-string character "\0".

Example:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
			DEVICE NAME														0 … 0			
Value	00	0E	8C	00	6D	75	73	65	5F	72	6F	62	65	72	74	6F				

- DEVICE NAME: “6D 75 73 65 5F 72 6F 62 65 72 74 6F” which corresponds to “muse_roberto”.

5.2.2 Device ID

The Device ID command can be executed in read mode only. The HEX code is 0x8E. The perform a get device ID operation, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	8E	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	06	8E	ERROR CODE	DEVICE ID				0 … 0											

- DEVICE ID
  It represents the unique identifier of the device, as a 32-bit unsigned integer that can be converted into the corresponding hex string representation.

Example:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
			DEVICE ID				0 … 0													
Value	00	06	8E	00	03	46	B5	83												

- DEVICE ID: “03 46 B5 83” which corresponds to “83B54603”. Bytes in reverse order to keep endianness w.r.t. the corresponding 32-bit unsigned integer numeric value.

5.2.3 Device Skills

The Device Skills command can be executed in read mode only. The HEX code is 0x8F. The perform a get device skills operation, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	8F	01	SKILLS CODE	0 … 0																

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	06	8F	ERROR CODE	SKILLS										0 … 0					

- SKILLS CODE
  It identifies the skill set of interest among:
  - HARDWARE SKILLS 0x00
  - SOFTWARE SKILLS 0x01

- SKILLS
  It is a 32-bit unsigned integer value that represents the required skill set in response buffer. While Software skills depends on the firmware and algorithm licenses installed, the numeric value corresponding to hardware skills must be decoded performing a bitwise operation using the following mask (Table 6).

HARDWARE	
Skill	**Code**
Gyroscope	0x0001
Accelerometer	0x0002
Magnetometer	0x0004
High Dynamic Range (HDR) Accelerometer	0x0008
Temperature	0x0010
Relative Humidity	0x0020
Ambient Barometric Pressure	0x0040
Light intensity (Visible Spectrum)	0x0080
Light intensity (IR)	0x0100
Range / distance	0x0200
Microphone	0x0400

TABLE 6: BIT-MASK FOR HARDWARE SKILLS DECODING.

5.2.4 Battery Charge and Voltage

The Battery Charge and Battery Voltage commands can be executed in read mode only. The HEX codes are: 0x87 (charge level), and 0x88 (voltage).

The access the current **battery charge** level, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	87	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	03	87	ERROR CODE	BATTERY CHARGE										0 … 0					

- BATTERY CHARGE
  It represents the battery charge level in percentage format, as an 8-bit unsigned integer.

The access the current **battery voltage** level, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	88	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	04	88	ERROR CODE	BATTERY VOLTAGE		0 … 0													

- BATTERY VOLTAGE
  It represents the battery voltage as a 16-bit unsigned integer value.

5.2.5 System State

The Muse v3 device behavior is controlled through a state machine (Figure 2) implemented at a firmware level. It depends on a set of pre-defined user scenarios or operating modes, as well as the functions supported by the device itself. The available states, that can be (directly and indirectly) controlled by the end-user are summarized in Table 7.

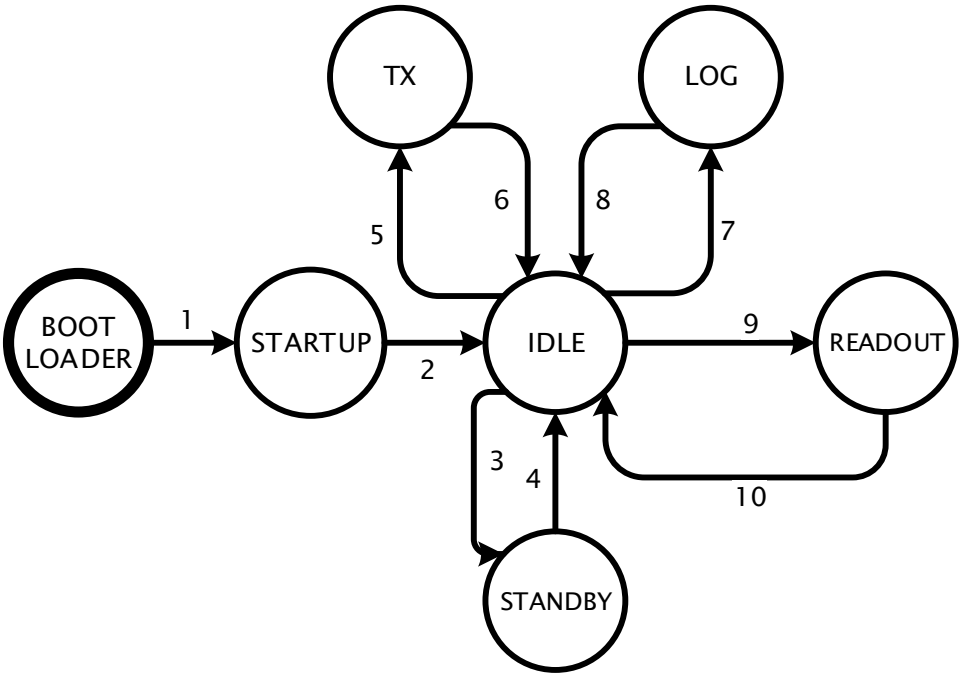

FIGURE 2: STATE MACHINE REPRESENTATION.

Name	Coding	Description
IDLE	0x02	System in IDLE state
STANDBY	0x03	System in STANDY state
LOG	0x04	System in LOG state
READOUT	0x05	System in the READOUT state
CALIBRATION	0x07	System in CALIBRATION state
TX	0x06	System in TX state – *BUFFERED MODE*
	0x08	System in TX state – *DIRECT MODE*

TABLE 7: SYSTEM STATES.

The system state command can be executed in both write and read mode. The HEX code is 0x02 (write), and 0x82 (read).

To **SET** a new system state, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	02	01	STATE TO BE SET	0 … 0																

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	02	02	ERROR CODE	0 … 0															

- STATE TO BE SET
  It must be filled with the HEX code of the state to be set.

To **GET** the current system state, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	82	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	03	82	ERROR CODE	SYSTEM STATE	0 … 0														

- SYSTEM STATE
  It provides the HEX code corresponding to the current state of the system.

The parameters required by each state are discussed in Section 5.4 (Data Acquisition, LOG and TX), and Section 5.6 (Memory, READOUT).

5.3 Settings

5.3.1 Full Scales

The full scales command can be executed in both write and read mode. The HEX code is 0x40 (write), and 0xC0 (read).

To **SET** a new full scales configuration, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	40	03	SENSORS FULL SCALE		0 … 0															

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	02	40	ERROR CODE	0 … 0															

- SENSORS FULL SCALE
  It is a 24-bit unsigned integer representation of the full scales to be set. It must be encoded in HEX format.
  Each sensor has 2-bits reserved for its full scale, in order: Gyroscope, Accelerometer, High Dynamic Range (HDR) Accelerometer, and Magnetometer.

```
/**
 * Structure containing the sensors full scale
 */
typedef struct {
        uint8_t gyro_fs :2;
        uint8_t axl_fs :2;
        uint8_t hdr_axl_fs :2;
        uint8_t magn_fs :2;
        uint16_t rfu;
} __attribute__((packed)) System_sensorsFS;
```

To generate the 24-bit unsigned integer value to be used as FULL-SCALE CODE, sensor configurations (Table 8) must be combined with a **bitwise OR**.

Sensor	**GYROSCOPE [dps]**				**ACCELEROMETER [g]**				**HDR ACCELEROMETER [g]**			**MAGNETOMETER [Gauss]**			
FS value	245	500	1000	2000	4	32	8	16	100	200	400	4	8	12	16
Sensitivity[1]	0.00875	0.0175	0.035	0.070	0.122	0.976	0.244	0.488	49	98	195	1000/ 6842	1000/ 3421	1000/ 2281	1000/ 1711
HEX code	0x00	0x01	0x02	0x03	0x00	0x04	0x08	0x0c	0x00	0x10	0x30	0x00	0x40	0x80	0xc0

TABLE 8: FULL SCALE CONFIGURATIONS.

[1] The sensitivity coefficients have the following unit of measure, in order: dps/LSB (Gyroscope), mg/LSB (Accelerometer), mg/LSB (HDR Accelerometer), mGauss/LSB (Magnetometer).

To **GET** the current full scales configuration, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	C0	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	05	C0	ERROR CODE	SENSORS FULL SCALE									0 … 0						

- SENSORS FULL SCALE
  It is a 24-bit unsigned integer representation of the current full scales configuration that must be decoded performing a **bitwise AND** using the following bit mask (Table 9), one for each sensor.

Sensor	**Mask Value**
Gyroscope	0x03
Accelerometer	0x0c
Magnetometer	0xC0
High Dynamic Range (HDR) Accelerometer	0x30

TABLE 9: BIT-MASK DEFINITION FOR FULL SCALE DECODING OPERATIONS.

Example:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
					SENSORS FULL SCALE			0 … 0												
Value	00	05	C0	00	0a	00	00	0 … 0												

- SENSORS FULL SCALE: “0A0000” from which the sensors configuration can be extracting as follows:

  1. Perform a bitwise AND using the masks provided in Table 8.
     With the code retrieved in this example, we obtained:
     - Gyroscope code: 0x02
     - Accelerometer code: 0x08
     - Magnetometer code: 0x00
     - HDR Accelerometer code: 0x00

  2. Retrieve the full scale and corresponding sensitivity coefficient from the configuration dictionary:
     - Gyroscope: 1000 / 0.035
     - Accelerometer: 8 / 0.244
     - Magnetometer 4 / 0.146156088
     - HDR Accelerometer: 100 / 49

5.3.2 User Configuration

The user can customize the Muse behaviors operating on a set of configuration flags, including:

- STANDBY
  It allows to enable (value: 1) or disable (value: 0) automatic standby. If enabled, the device will move to standby condition after 120 seconds of inactivity. The default value is ENABLED (value: 1). The system can also be forced into standby by using the button when the automatic standby is disabled.
  If the device is connected, the system is prevented from going into standby condition.

- CIRCULAR MEMORY
  It allows to enable (value: 1) or disable (value: 0) memory management using a circular buffer instead of a linear one. The default value is DISABLED (value: 0). If enabled, the memory is treated as a circular buffer, which means that when the memory runs out, the oldest data is deleted to make space for new. This means that a log can never be interrupted by an end-of-memory condition.

A memory erase is carried out whenever this command is used to switch memory mode, which can lead to data loss. ***Users are encouraged to save all files before switching memory behavior.***

- STREAMING CHANNEL
  It allows to enable (value: 1) or disable (value: 0) the streaming over USB operating mode. The default value is DISABLED (value: 0). If enabled, it indicates that the streaming is performed via USB instead of BLE.

With ***Muse_v3_WiFi-enabled*** *version, the STREAM CHANNEL register also accepts the values related to WiFi communication. Please refer to* ***AN_221e_Muse_v3_wifi_expansion_vx.x*** *for a detailed description.*

The user configuration command can be executed in both write and read mode. The HEX code is 0x51 (write), and 0xd1 (read).

To **SET** a new user configuration, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	51	04	BIT MASK		USER CONFIGURATION		0 … 0													

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	02	51	ERROR CODE	0 … 0															

- BIT MASK
  It is a 16-bit unsigned integer value representing the configuration channels to be set. See Table 10.

- USER CONFIGURATION
  It is a 16-bit unsigned integer value representing the configuration to be set.

Configuration	Mask Value
Standby (1 bit)	0x01
Circular Memory (1 bit)	0x02
Streaming Channel (3 bits)	0x1c

TABLE 10: BIT-MASK DEFINITION FOR USER CONFIGURATION MANAGEMENT.

To properly encode all the accepted configuration, the following tables summarize the bit order and hex/decimal codes to be used.

	LSB Byte								MSB Byte							
Bit Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Field	STANDBY	CIRCULAR MEMORY	CHANNEL			RFU (0xff)										

TABLE 11: USER CONFIGURATION REGISTER.

STANDBY		CIRCULAR MEMORY		CHANNEL		USER CONFIGURATION
Status	*Value*	*Status*	*Value*	*Status*	*Value*	
False	0	False	0	False	0	0
True	1	False	0	False	0	1
False	0	True	2	False	0	2
True	1	True	2	False	0	3
False	0	False	0	True	4	4
True	1	False	0	True	4	5
False	0	True	2	True	4	6
True	1	True	2	True	4	7

TABLE 12: CODES TO BE SET AS USER CONFIGURATION BASED ON ENABLED / DISABLED FLAGS.

To **GET** current user configuration, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	D1	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	04	D1	ERROR CODE	USER CONFIGURATION		0 … 0													

- USER CONFIGURATION
  It is a 16-bit unsigned integer value representing the current configuration. To decode the value of each configuration flag, a **bitwise AND** operation using the masks provided in Table 10 must be performed.

5.3.3 Calibration (MPE required)

All the calibration procedures, independently from the sensors to be calibrated, must be started using the set state command to enter the calibration state (HEX code: 0x07), providing the calibration type (i.e., which identify the sensor to be calibrated) as first argument and the field intensity value to fit (i.e., a 4-bytes floating point value) as second input parameter.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	02	06	07	CALIBRATION TYPE	FIELD INTENSITY				0 … 0											

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	02	02	ERROR CODE	0 … 0															

- CALIBRATION TYPE
  Define what MEMS is to be calibrated.
  1. Accelerometer 0x00
  2. Gyroscope 0x01
  3. Magnetometer 0x02

- FIELD INTENSITY
  It is a 32-bit floating point representing the field to be fit by the calibration routine.
  1. Accelerometer 1000 (HEX: 0x447a0000, that must be transmitted LSB)
  2. Gyroscope 0
  3. Magnetometer 400.0 (HEX: 0x43c80000, that must be transmitted LSB)

GYROSCOPE

The gyroscope calibration entails leaving the system still for the duration of the procedure. The aim is to estimate gyroscope offset around each axis. The calibration start is advertised by the flashing of green LED. The system must be left still during the data sampling.

ACCELEROMETER

The calibration procedure for the accelerometer entails sampling the data of a still system in 14 different positions. The calibration procedure starts with the first position. The system should be left still during the data sampling, signaled by fast flashing of the green LED. The sampling of the next position can be started by sending an ACK for the set state command, or the calibration procedure can be started with the data sampled so far by sending a NACK for the set state command.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	03	02	00	07	0 … 0														
	ACK	PAYLOAD LENGTH	COMMAND	ERROR CODE	CALIBRATION STATE															

As stated above, by sending the provided ACK, the system will start a new sampling phase, so the user should make sure to position the system correctly before sending the ACK. The system should remain still for the entire duration of the sampling phase (green led flashing fast) and can be moved to allow for re-positioning while the green led is flashing slowly. Once the sampling phase is completed (i.e., 14 different positions is the suggested method) the user should send a NACK to start the calibration algorithm based on previously sampled data.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	03	02	01	07	0 … 0														
	ACK	PAYLOAD LENGTH	COMMAND	END OF CALIBRATION	CALIBRATION STATE															

MAGNETOMETER

The magnetometer calibration entails performing as many different system rotations as possible during the procedure.

The magnetometer sensor can be greatly influenced by nearby interferences, coming from nearby electronic devices or appliances, or simply from ferromagnetic materials. The calibration should be performed in an interference free ambient, thus the user should perform the calibration standing in the middle of a room, far from other electronic devices, if possible.

5.4 Data Acquisition

Data acquisition is controlled by using the set state command performed on Command Characteristic. Based on the selected acquisition channel (i.e., STREAM or LOG, independently from the selected acquisition mode), the device starts to stream the data through notification on Data Characteristic or save the data on the flash memory, respectively. The general format of the command is as follows, independently from the sensor configuration, state, and selected acquisition mode or frequency.

To **START** data acquisition, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	02	05	STATE	ACQUISITION MODE			ACQUISITION FREQUENCY		0 … 0											

RESPONSE

Index	0	1	2	3	4 … 6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**															
Value	00	09	02	ERROR CODE	SENSORS FULL SCALE		ACQUISITION MODE		ACQUISITION FREQUENCY	0 … 0								

In this case, the response is a confirmation message including the full-scale configuration currently set as well as the selected acquisition mode and frequency.

- ACQUISITION MODE
  It is the 24-bit mask representing the acquisition mode currently set. It determines the set of data read from the device.

- ACQUISITION FREQUENCY
  It is the 8-bit enum representing the desired acquisition frequency currently set. The acquisition will be performed at this frequency.

- SENSORS FULL SCALE
  It is an 8-bit unsigned integer representation of the current full scales configuration that must be decoded performing a **bitwise AND** using the masks provided in Table 8.

*The system may **fail to execute a start acquisition** command in the following cases:*

- *The memory is being erased.*
- *The content of the payload received is not correct.*
- *A system restart has been scheduled.*
- *The battery is too low (< 3300 mV).*
- *The system is not in IDLE state.*

Mode	Description	Code (hex)	Size [bytes]
GYR	Gyroscope	0x000001	6
AXL	Accelerometer	0x000002	6
IMU	Inertial Measurement Unit (IMU, Gyroscope + Accelerometer)	0x000003	12
MAG	Magnetometer	0x000004	6
9DOF	9 Degrees Of Freedom (DOFs, Gyroscope + Accelerometer + Magnetometer)	0x000007	18
HDR	High Dynamic Range (HDR) Accelerometer	0x000008	6
IMU / HDR	IMU + HDR	0x000011	18
QUAT	Orientation Quaternion	0x000010	6
TIME	Timestamp	0x000020	6
TEMP / HUM	Temperature + Humidity	0x000040	6
TEMP / PRESS	Temperature + Barometric Pressure	0x000080	6
RANGE	Range + Light Intensity	0x000100	6
SOUND	Microphone	0x000400	6

TABLE 13: ACQUISITION MODES DEFINITION.

The communication protocol provides a predefined set of acquisition modes which can be combined as you like, paying attention to only one rule: ***the total size of the package must be equal to 6, 12, 24, 30, or 60 bytes****. Any other combination of size different from those listed here will return an acknowledge error. For example,* in Table 13 *9DOF and IMU/HDR modes cannot be used alone They must be associated with another data, for example Timestamp, to have an overall admitted packet size.*

Example of packet dimension computation, given a specific acquisition mode combination:

```
public static int GetPacketDimension(DataMode inMode)
{
    int packet_dimension = 0;

    // Compute packet dimension given data acquisition mode
    if ((inMode & DataMode.DATA_MODE_GYRO) > 0)
        packet_dimension += (int)DataSize.DATA_SIZE_GYRO;
    if ((inMode & DataMode.DATA_MODE_AXL) > 0)
        packet_dimension += (int)DataSize.DATA_SIZE_AXL;
    if ((inMode & DataMode.DATA_MODE_HDR) > 0)
        packet_dimension += (int)DataSize.DATA_SIZE_HDR;
    if ((inMode & DataMode.DATA_MODE_MAGN) > 0)
        packet_dimension += (int)DataSize.DATA_SIZE_MAGN;
    if ((inMode & DataMode.DATA_MODE_ORIENTATION) > 0)
        packet_dimension += (int)DataSize.DATA_SIZE_ORIENTATION;
    if ((inMode & DataMode.DATA_MODE_TIMESTAMP) > 0)
        packet_dimension += (int)DataSize.DATA_SIZE_TIMESTAMP;
    if ((inMode & DataMode.DATA_MODE_TEMP_HUM) > 0)
        packet_dimension += (int)DataSize.DATA_SIZE_TEMP_HUM;
    if ((inMode & DataMode.DATA_MODE_TEMP_PRESS) > 0)
        packet_dimension += (int)DataSize.DATA_SIZE_TEMP_PRESS;
    if ((inMode & DataMode.DATA_MODE_RANGE) > 0)
        packet_dimension += (int)DataSize.DATA_SIZE_RANGE;
    if ((inMode & DataMode.DATA_MODE_SOUND) > 0)
        packet_dimension += (int)DataSize.DATA_SIZE_SOUND;

    // Check packet dimension consistency
    // It MUST be a dividend of 120 (2040)
    if (packet_dimension == 6 || packet_dimension == 12 || packet_dimension == 24 ||
            packet_dimension == 30 || packet_dimension == 60)
        return packet_dimension;

    // Return -1 in case of data dimension inconsistency
    return -1;
}
```

Frequency [Hz]	Code (hex)
25	0x01
50	0x02
100	0x04
200	0x08
400	0x10
800	0x20
1600	0x40

TABLE 14: ACQUISITION FREQUENCYS.

To **STOP** data acquisition, the following command must be composed to put the system in **IDLE (0x02) state**.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	02	01	02	0 … 0																

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	02	02	ERROR CODE	0 … 0															

5.4.1 Acquisition Modes

Different combination of the above fields allows to perform data acquisition in a few different operating modes:

- **STREAMING**
  The device is connected to a master via BLE, typically desktop or mobile application, and performs a real-time streaming of the data based on the selected acquisition mode and frequency.

1. **BUFFERED**
   Data is arranged within the 128 bytes of the Data characteristic according to the packet dimension based on the selected acquisition mode. The notification frequency is lower than the acquisition frequency because the data is pooled within the Data characteristic.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	TYPE	LENGTH	VALUE (Payload)																	
Value	02	05	**06**	ACQUISITION MODE			ACQUISITION FREQUENCY		0 … 0											

RESPONSE

Index	0	1	2	3	4 … 6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	TYPE	LENGTH	VALUE (Payload)															
Value	00	09	02	ERROR CODE	SENSORS FULL SCALE		ACQUISITION MODE		ACQUISITION FREQUENCY	0 … 0								

Example of number of packets computation given a specific data acquisition mode combination:

```
public static int GetNumberOfPackets(DataMode mode)
{
    // Compute packet dimension and check its consistency given data acquisition mode
    int packet_dimension = GetPacketDimension(mode);

    // Compute overall number of packets contained into a data characteristic update
    if (packet_dimension != -1)
        return (120 / packet_dimension);

    // Return -1 in case of data dimension inconsistency
    return -1;
}
```

2. **DIRECT**
   Data is arranged within the 128 bytes of the Data characteristic one packet at a time. The notification and acquisition frequency match.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	TYPE	LENGTH	VALUE (Payload)																	
Value	02	05	**08**	ACQUISITION MODE			ACQUISITION FREQUENCY		0 … 0											

RESPONSE

Index	0	1	2	3	4 … 6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	TYPE	LENGTH	VALUE (Payload)															
Value	00	09	02	ERROR CODE	SENSORS FULL SCALE		ACQUISITION MODE		ACQUISITION FREQUENCY	0 … 0								

In this case it is not necessary to compute the overall number of packets: there is one packet for each notification.

- **LOG**

  The device can be connected to a master via BLE or can be controlled directly from the smart button, if properly configured. The data is acquired and saved into the device flash memory.

  1. **COMMAND**

     The log acquisition can be controlled (i.e., start, stop) using the set state command.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	02	05	**04**	ACQUISITION MODE			ACQUISITION FREQUENCY		0 … 0											

RESPONSE

Index	0	1	2	3	4 … 6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**															
Value	00	09	02	ERROR CODE	SENSORS FULL SCALE		ACQUISITION MODE		ACQUISITION FREQUENCY	0 … 0								

  2. **BUTTON**

     If properly configured (see Section 5.4.2 for further details), the acquisition can be started and stopped with a single click on the system button.

5.4.2 Button Log Configuration

The button log configuration command can be executed in both write and read mode. The HEX code is 0x50 (write), and 0xD0 (read).

To **SET** a new button log configuration, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	50	04	ACQUISITION MODE			ACQUISITION FREQUENCY	0 … 0													

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	02	50	ERROR CODE	0 … 0															

To **GET** the current button log configuration, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	D0	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	06	D0	ERROR CODE	ACQUISITION MODE			ACQUISITION FREQUENCY	0 … 0											

- ACQUISITION MODE
  It is the 24-bit mask representing the acquisition mode to be set (write) or currently set (read) for log operations. It must be encoded in HEX format.

- ACQUISITION FREQUENCY
  It is the 8-bit enum representing the desired acquisition frequency to be set (write) or currently set (read) for log operations.

Example:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
					ACQUISITION MODE			ACQUISITION FREQUENCY	0 … 0											
Value	00	06	D0	00	27	00	00	08												

- ACQUISITION MODE: “27 00 00” which corresponds to 39 as 32-bit unsigned integer. The corresponding data acquisition mode combination can be extracted by performing a **bitwise AND** operation using the hex code of Table 13.

- ACQUISITION FREQUENCY: “08” which corresponds to 200 Hz. See Table 14.

5.5 Data Decoding

5.5.1 Gyroscope: 3D Angular Rate

The 3D angular rate is provided as a raw value read out from the sensor registers, represented as a two's complement, Little Endian encoded, 16-bit signed integer value. The conversion from raw value to actual measurement must be performed on the master side considering the actual full-scale set and the corresponding sensitivity coefficient (see Table 8).

3D Gyroscope [LSB]					
x		y		z	
LSB	MSB	LSB	MSB	LSB	MSB

The output measure is provided in Degrees Per Second, dps.

Each channel (i.e., x, y, z) must be converted from raw 2-bytes array to the corresponding 16-bit signed integer representation (i.e., Int16) and then scaled to the actual metric representation multiplying it by the appropriate sensitivity coefficient.

Example of decoding implementation:

```
private static float[] DataTypeGYR(byte[] currentPayload, float gyr_res)
{
    // Define 3-elements array of float (i.e., x, y, z channels)
    float[] currentData = new float[3];

    // Define temporary internal variable used for data conversion
    byte[] tmp = new byte[2];

    // Iterate across Gyroscope channels
    for (int i = 0; i < 3; i++)
    {
        // Extract channel raw value (i.e., 2-bytes each)
        Array.Copy(currentPayload, 2*i, tmp, 0, 2);

        // Convert to Int16 and apply sensitivity scaling
        currentData[i] = (float)(BitConverter.ToInt16(tmp, 0) * gyr_res);
    }

    // Return decoded Gyroscope reading
    return currentData;
}
```

5.5.2 Accelerometer: 3D Linear Acceleration

The 3D acceleration is provided as a raw value read out from the sensor registers, represented as a two's complement, Little Endian encoded, 16-bit signed integer value. The conversion from raw value to actual measurement must be performed on the master side considering the actual full-scale set and the corresponding sensitivity coefficient (see Table 8).

3D Accelerometer [LSB]					
x		y		z	
LSB	MSB	LSB	MSB	LSB	MSB

The output measure is provided in millig, mg.

Each channel (i.e., x, y, z) must be converted from raw 2-bytes array to the corresponding 16-bit signed integer representation (i.e., Int16) and then scaled to the actual metric representation multiplying it by the appropriate sensitivity coefficient.

Example of decoding implementation:

```
private static float[] DataTypeAXL(byte[] currentPayload, float axl_res)
{
    // Define 3-elements array of float (i.e., x, y, z channels)
    float[] currentData = new float[3];

    // Define temporary internal variable used for data conversion
    byte[] tmp = new byte[2];

    // Iterate across Accelerometer channels
    for (int i = 0; i < 3; i++)
    {
        // Extract channel raw value (i.e., 2-bytes each)
        Array.Copy(currentPayload, 2*i, tmp, 0, 2);

        // Convert to Int16 and apply sensitivity scaling
        currentData[i] = (float)(BitConverter.ToInt16(tmp, 0) * axl_res);
    }

    // Return decoded Accelerometer reading
    return currentData;
}
```

5.5.3 Magnetometer: 3D Magnetic Field Intensity

The 3D angular rate is provided as a raw value read out from the sensor registers, represented as a two's complement, Little Endian encoded, 16-bit signed integer value. The conversion from raw value to actual measurement must be performed on the master side considering the actual full-scale set and the corresponding sensitivity coefficient (see Table 8).

3D Magnetometer [LSB]					
x		y		z	
LSB	MSB	LSB	MSB	LSB	MSB

The output measure is provided in milli-Gauss, mGauss.

Each channel (i.e., x, y, z) must be converted from raw 2-bytes array to the corresponding 16-bit signed integer representation (i.e., Int16) and then scaled to the actual metric representation multiplying it by the appropriate sensitivity coefficient. It is worth noting that the sensitivity of the magnetometer depends on the hardware revision of the user board. In the first hardware revision the sensitivity of the sensor is a fixed 1.5 mGauss/LSB.

Example of decoding implementation:

```
private static float[] DataTypeMAG(byte[] currentPayload, float mag_res)
{
    // Define 3-elements array of float (i.e., x, y, z channels)
    float[] currentData = new float[3];

    // Define temporary internal variable used for data conversion
    byte[] tmp = new byte[2];

    // Iterate across Magnetometer channels
    for (int i = 0; i < 3; i++)
    {
        // Extract channel raw value (i.e., 2-bytes each)
        Array.Copy(currentPayload, 2*i, tmp, 0, 2);

        // Convert to Int16 and apply sensitivity scaling
        currentData[i] = (float)(BitConverter.ToInt16(tmp, 0) * mag_res);
    }

    // Return decoded Magnetometer reading
    return currentData;
}
```

5.5.4 High Dynamic Range (HDR) Accelerometer: 3D Linear Acceleration

The 3D HDR acceleration is provided as a raw value read out from the sensor registers, represented as a two's complement, left justified, Little Endian encoded, 12-bit signed integer value. The conversion from raw value to actual measurement must be performed on the master side considering the actual full-scale set and the corresponding sensitivity coefficient (see Table 8).

3D HDR Accelerometer [LSB]					
x		y		z	
LSB	MSB	LSB	MSB	LSB	MSB

Each channel (i.e., x, y, z) must be converted from raw 2-bytes array to the corresponding 16-bit signed integer representation (i.e., Int16) and then scaled to the actual metric representation multiplying it by the appropriate sensitivity coefficient. It is worth noting that the sensitivity provided in Table 8 does not account for the 4 bits right shift operation which shall be carried out to account for the 12-bit resolution of the sensor. This means that the integer value must be divided by 16 before sensitivity scaling.

Example of decoding implementation:

```
private static float[] DataTypeHDR(byte[] currentPayload, float hdr_res)
{
    // Define 3-elements array of float (i.e., x, y, z channels)
    float[] currentData = new float[3];

    // Define temporary internal variable used for data conversion
    byte[] tmp = new byte[2];

    // Iterate across HDR channels
    for (int i = 0; i < 3; i++)
    {
        // Extract channel raw value (i.e., 2-bytes each)
        Array.Copy(currentPayload, 2*i, tmp, 0, 2);

        // Convert to Int16 and apply resolution and sensitivity scaling
        currentData[i] = (float)((BitConverter.ToInt16(tmp, 0) / 16) * hdr_res);
    }

    // Return decoded HDR reading
    return currentData;
}
```

5.5.5 Orientation Quaternion (MPE required)

The orientation is presented as a normalized quaternion, but only the three imaginary components are reported in the data, 2 bytes each. Each reported component is transmitted as a signed integer, mapped inside the range [-1,1], meaning that the LSB value is $\frac{1}{2^{15}-1}$. From this, the user should read the 2 bytes corresponding to the quaternion component as a signed 16-bit integer, then divide it by 32767 to obtain the float equivalent (with the approximations intrinsic to the method). The user can then use the knowledge that the quaternion is normalized, and recalculate the real component as

$$q_0 = \sqrt{1 - q_1^2 + q_2^2 + q_3^2}$$

Since this method drops the real part of the quaternion, given a quaternion **q:**

$$q = q_0 + \boldsymbol{i}q_1 + \boldsymbol{j}q_2 + \boldsymbol{k}q_3 | \, q_0 < 0$$

Imaginary components of quaternion **q** are transmitted as $-q_1$, $-q_2$ and $-q_3$, so that the reconstructed quaternion is

$$q_{reconstructed} = q_0 - \boldsymbol{i}q_1 - \boldsymbol{j}q_2 - \boldsymbol{k}q_3 = q$$

Orientation Quaternion (AHRS)					
i		j		k	
LSB	MSB	LSB	MSB	LSB	MSB

Example of decoding implementation:

```
private static float[] DataTypeOrientation(byte[] currentPayload)
{
    // Define 4-elements array of float (i.e., qw, qi, qj, qk channels)
    float[] currentData = new float[4];

    // Define temporary internal variable used for data conversion
    byte[] tmp = new byte[2];

    // Orientation Quaternion (i.e., UNIT QUATERNION)
    float[] tempFloat = new float[3];
    for (int i = 0; i < 3; i++)
    {
        // Extract channel raw value (i.e., 2-bytes each) and convert to Int16
        tempFloat[i] = BitConverter.ToInt16(currentPayload, 2*i);

        // Keep into account the data resolution
        tempFloat[i] /= 32767;

        // Assign imaginary parts of quaternion
        currentData[i + 1] = tempFloat[i];
    }

    // Compute real component of quaternion given immaginary parts
    currentData[0] = (float)Math.Sqrt(1 - (currentData[1] * currentData[1] +
            currentData[2] * currentData[2] + currentData[3] * currentData[3]));

    // Return decoded Unit Quaternion
    return currentData;
}
```

5.5.6 Timestamp

The sub-second timestamp reports the number of milliseconds since 26 January 2020 00:53:20. The user can reconstruct the Unix-style epoch by adding the epoch of the date (in milliseconds), **1580000000**.

Time					
LSB (B0)	B1	B2	B3	B4	MSB (B5)

Example of decoding implementation:

```
private static ulong DataTypeTimestamp(byte[] currentPayload)
{
    // Set current data variable to 0
    ulong currentData = 0;

    // Get raw byte array representation to be decoded
    byte[] tmp = new byte[8];
    Array.Copy(currentPayload, 0, tmp, 0, 6);

    // Convert raw data to 64-unsigned integer representation
    UInt64 tempTime = (ulong)BitConverter.ToUInt64(tmp, 0);
    tempTime &= 0x0000FFFFFFFFFFFF;

    // Add reference epoch
    tempTime += ((UInt64)REFERENCE_EPOCH) * 1000;

    currentData = tempTime;

    // Return current date/time
    return currentData;
}
```

5.5.7 Temperature and Relative Humidity

The temperature and relative humidity are sampled at a fixed rate of 25 Hz. The user can reconstruct the values by converting the provided byte array into an unsigned integer and using the following equations:

$$T = Data\_array_T \cdot 0.002670 - 45$$

$$RH = Data\_array_{RH} \cdot 0.001907 - 6$$

The user should also note that, to keep the data size to the constant 6-bytes value, 2 padding zeros are added after the relative humidity data.

Temp & Relative Humidity					
Temp.		**RH**		**Padding**	
LSB	MSB	LSB	MSB	0	0

5.5.8 Temperature and Ambient Barometric Pressure

The temperature and ambient pressure are sampled at a fixed rate of 25 Hz. The user can reconstruct the values by converting the provided byte array into an unsigned integer and using the following equations:

$$P = \frac{Data\_array_{PRESS}}{4096}$$

$$T = \frac{Data\_array_T}{100}$$

The user should also note that, to keep the data size to the constant 6-bytes value, one padding zero is added after the temperature data.

Temp & Ambient Pressure					
Press			**Temp**		**Pad**
LSB	…	MSB	LSB	MSB	0

5.5.9 Range and Light Intensity

The range, visible light intensity and infrared light intensity are sampled at a fixed rate of 25 Hz.

Range					
Range		**LUM**		**IR LUM**	
LSB	MSB	LSB	MSB	LSB	MSB

The user can reconstruct the true ambient light intensity in lumens via a spline fit, as follows:

- The user should identify the reported value of the visible light and infrared light by converting the reported values to 16-bit unsigned integers. For the sake of this chapter, the two converted values will be referred as LUM_{VIS} and LUM_{IR}.

- The user should then identify the overall light source type using the ratio between LUM_{VIS} and LUM_{IR}:
  - CASE 1: $\frac{LUM_{IR}}{LUM_{VIS}} < 0.109$ covers LEDs, fluorescence, and sunlight
  - CASE 2: $\frac{LUM_{IR}}{LUM_{VIS}} < 0.429$ covers incandescent and halogen lamps
  - CASE 3: $\frac{LUM_{IR}}{LUM_{VIS}} < (0.95\backslash cdot\ 1.45)$ covers dimmed incandescent and halogen lamps
  - CASE 4: $\frac{LUM_{IR}}{LUM_{VIS}} < (1.5\backslash cdot\ 1.45)$ covers dimmed incandescent and halogen lamps
  - CASE 5: $\frac{LUM_{IR}}{LUM_{VIS}} < (2.5\backslash cdot\ 1.45)$
  - CASE 6: all other cases

- Once identified the case, the true lumens value can be calculated with the following equations:
  - CASE 1: $LUX = 1.534 \cdot LUM_{VIS} - 3.759 \cdot LUM_{IR}$
  - CASE 2: $LUX = 1.339 \cdot LUM_{VIS} - 1.972 \cdot LUM_{IR}$
  - CASE 3: $LUX = 0.701 \cdot LUM_{VIS} - 0.483 \cdot LUM_{IR}$
  - CASE 4: $LUX = 2 \cdot 0.701 \cdot LUM_{VIS} - 1.18 \cdot 0.483 \cdot LUM_{IR}$
  - CASE 5: $LUX = 4 \cdot 0.701 \cdot LUM_{VIS} - 1.33 \cdot 0.483 \cdot LUM_{IR}$
  - CASE 6: $LUX = 8\backslash cdot\ 0.701 \cdot LUM_{VIS}$

In summary, the user should implement the following logic:

```
if (ir / vis < 0.109f) {
        lux = 1.534f * vis - 3.759f * ir;
} else if (ir / vis < 0.429f) {
        lux = 1.339f * vis - 1.972f * ir;
} else if (ir / vis < 0.95f * 1.45f) {
        lux = 0.701f * vis - 0.483f * ir;
} else if (ir / vis < 1.5f * 1.45f) {
        lux = 2.0f * 0.701f * vis - 1.18f * 0.483f * ir;
} else if (ir / vis < 2.5f * 1.45f) {
        lux = 4.0f * 0.701f * vis - 1.33f * 0.483f * ir;
} else {
        lux = 8.0f * 0.701f * vis;
}
```

5.6 Memory

5.6.1 Control

The memory control command can be executed in both write and read mode. The HEX code is 0x20 (write), and 0xA0 (read).

To perform an **ERASE MEMORY** operation, the command must be executed in WRITE mode. The following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	20	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	03	20	ERROR CODE	MEMORY ERASE STATUS	0 … 0														

- MEMORY ERASE STATUS
  It provides feedback on memory erase process status. It can be one of the following values:
  - 0x01, erase memory correctly scheduled.
  - 0x02, erase memory completed.

The user should expect two ACKs, the first containing the MEMORY ERASE STATUS code 0x01, once the procedure has started, and a second one containing code 0x02 once the procedure has been completed. The entire memory erase should take around 3 seconds to complete.

To **GET** the current memory status in terms of available space (i.e., in percentage format) and number of files currently saved in memory, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	A0	00	0 … 0																	

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	05	A0	ERROR CODE	AVAILABLE SPACE	NUMBER OF FILES														

- AVAILABLE SPACE
  Available space, provided in percentage format (i.e., 8-bit unsigned integer value).

- NUMBER OF FILES
  It is the 16-bit unsigned integer representing the number of files saved in memory.

5.6.2 File Information

The file information command can be executed in read-only mode. The HEX code is 0xA1.

To **GET** the file info about a specific file identifier, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	A1	02	FILE ID		0 … 0															

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	0C	A1	ERROR CODE	TIMESTAMP					SENSORS FULL SCALES	DATA ACQUISITION MODE			DATA ACQUISITION FREQUENCY	0 … 0					

- FILE ID
  The files are retrieved passing an index (i.e., between 0 and N-1, where N is the number of files currently stored in the system) as the command argument. If the requested file is present, the command will respond with the following information, in order:

  - TIMESTAMP
    Start timestamp of the file (i.e., 5 bytes).
    It must be decoded by converting the raw values into a 64-bit unsigned integer, and adding the REFERENCE_EPOCH = 1580000000[2], multiplied x1000 to keep into account the seconds resolution.

  - SENSORS FULL SCALES
    The sensors full scale code (i.e., 1 byte). It must be decoded performing a **bitwise AND** using the bit mask defined in Table 9.

  - DATA ACQUISITION MODE
    The data mode (i.e., 3 bytes).
    It must be decoded by converting the raw values into a 32-bit unsigned integer value and performing a **bitwise AND** using the bit mask defined in Table 9.

  - DATA ACQUISITION FREQUENCY
    The frequency of log for the file (i.e., 1 byte, the hex code representing the selected frequency).

Otherwise, an acknowledge message with ERROR CODE = 0x01 is returned.

The files stored in memory are numbered from 0 to N-1.

It is worth noting that only in this case the SENSORS FULL SCALES field is represented as an 8-bit unsigned integer code, instead of 24-bits like in data acquisition.

[2] It corresponds to Sunday 26 January 2020 00:53:20.

5.6.3 File Download

The file download command can be executed in write-only mode. The HEX code is 0x22.

To start the file download procedure given a specific file identifier, the following command must be composed.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	22	03	FILE ID		CHANNEL	0 … 0														

RESPONSE

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	06	22	ERROR CODE	FILE SIZE				0 … 0											

- FILE ID
  The files are retrieved passing an index (i.e., between 0 and N-1, where N is the number of files currently stored in the system) as the command argument.

- CHANNEL
  It allows to select the communication channel to be used (i.e., USB, 0x00, or BLE, 0x01).

- FILE SIZE
  It is a 32-bit unsigned integer value that represents the file size in bytes.

If all run successfully, the system moves into the READOUT state. The user MUST send an ACK message to effectively start the data streaming.

TRANSMISSION

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Field	**TYPE**	**LENGTH**	**VALUE (Payload)**																	
Value	00	02	22	00	0 … 0															

The ack sent by the user follows the same ack format of every other command: the first byte of the payload is used to indicate the command that triggered the ack (0x22 in this case), and the second byte is used to signify an ACK (0x00) or NACK (any other number).

This operation MUST be performed at the end of each page (2048 bytes), to signal the device that the page has been correctly received and the next page can be sent. In the case of NACK, the device attempts to re-send the same page. Once the last page has been received, the final acknowledge message will enable the system to return to IDLE status.

Concerning USB offload, files are sent at chunks of 1 memory page (i.e., 2048 bytes) at a time, and all data has a 4 bytes trailer "!!!!" appended.

In the case of BLE channel selection, the file will be transmitted 128 bytes at a time through notification on Data Characteristic. Every 2048 bytes received (i.e., 16 notifications), an ACK message is required. The flow is mostly the same as the USB, but carried out via Bluetooth: the firmware will move into the readout state, waiting for an ack before starting the data offload. After the first ack is received, the firmware will perform unconfirmed notifications (i.e., 128 bytes) to transmit the page (i.e., 2048 bytes), then will wait for a user ACK. As before, if the user sends a NACK, the same page will be retransmitted; otherwise, the next page will begin transmission with the same procedure. The last page will probably be completely transmitted with less than 16 notifications, but the user must send the ack at the end of the process anyway to return the system in the idle state.