iCOMOX Data Acquisition Kit

2.0

API User Guide

Table of Contents

1. Introduction

The iCOMOX is the central component of the iCOMOX Data Acquisition (DA) kit. Its purpose is to acquire information through its sensors, for the kit's host.

The iCOMOX is controlled via an API which is described in this document. The API commands are described here as C structs, which are exchanged between the iCOMOX to the PC that controls it.

Different interfaces can add more complexity to the layers below the API (frame sync header, fragmentation control, etc.), in order to support its transfer over the respective interface. These interfaces are not described in this document.

All of the structs are packed so that there are no "memory holes". Structs size exactly equals the size of their respected fields.

Unless explicitly mentioned, all of the fields use a little-endian format.

The messages structs names use the prefix of sCOMOX_IN_MSG_xxx and sCOMOX_OUT_MSG_xxx. For all the interfaces, the "IN" messages mean that the direction is to user side while the "OUT" messages mean that the direction is to the iCOMOX side.

The iCOMOX always acknowledges any OUT message that it receives with a respective IN message; however, the opposite is not supported.

2. API Description

2.1. List of Message Codes

All messages begin with a “Code” field, which indicates the type of transmitted message. A pair of OUT and IN message always share the same value of their “Code” field.

The enum definition of the “Code” field type (eCOMOX_MSG_CODE) is displayed below:

```
typedef enum __attribute__((__packed__))
{
	cCOMOX_MSG_CODE_Hello,
	cCOMOX_MSG_CODE_Reset,
	cCOMOX_MSG_CODE_GetConfiguration,
	cCOMOX_MSG_CODE_SetConfiguration,
	cCOMOX_MSG_CODE_Report,

	cCOMOX_MSG_CODE_COUNT
} eCOMOX_MSG_CODE;
```

2.2. Hello Messages

2.2.1. sCOMOX_IN_MSG_Hello

```
typedef struct __attribute__((__packed__))
{
        eCOMOX_MSG_CODE     Code;
        char                ID[6];     // = "iCOMOX"
        eCOMOX_BOARD_REV    BoardRevision;
        uint64_t            SerialNumber;
        sDATE_TIME          BuildVersion;
} sCOMOX_IN_MSG_Hello;
```

Field name	Field Size (bytes)	Description
Code	1	cCOMOX_MSG_CODE_Hello message type code.
ID	6	6 bytes that contain the following characters: “iCOMOX”.
BoardRevision	1	*cCOMOX_BOARD_REV_C constant, to indicate the current board revision.*
SerialNumber	8	A unique serial number of the iCOMOX.
BuildVersion	7	The exact date and time in which the firmware was compiled.

The sCOMOX_IN_MSG_Hello message is automatically sent by the iCOMOX after detecting a connection. It provides the host with all of the relevant version information that is needed, in order to determine the validity of the API.

The definition of the non-standard types, that are used inside, are displayed below:

```
typedef struct __attribute__((__packed__))
{
	uint16_t Year;		// 2019
	uint8_t		Month;		// 1-12
	uint8_t		DayInMonth;	// 1-31
} sDATE;
typedef struct __attribute__((__packed__))
{
	uint8_t		Hour;		// 0-23
	uint8_t		Minute;		// 0-59
	uint8_t		Second;		// 0-59
} sTIME;
typedef struct __attribute__((__packed__))
{
	sDATE		Date;
	sTIME		Time;
} sDATE_TIME;
typedef enum __attribute__((__packed__))
{
	cCOMOX_BOARD_REV_A,
	cCOMOX_BOARD_REV_B,
	cCOMOX_BOARD_REV_C,

	cCOMOX_BOARD_REV_COUNT
} eCOMOX_BOARD_REV;
typedef struct __attribute__((__packed__))
{
	uint8_t			Major;
	eCOMOX_BRANCH	Branch;
	uint8_t			Minor;
} sCOMOX_VERSION;
```

2.2.2. sCOMOX_OUT_MSG_Hello

```
typedef struct __attribute__((__packed__))
{
        eCOMOX_MSG_CODE                 Code;
} sCOMOX_OUT_MSG_Hello;
```

Field name	Field size (bytes)	Description
Code	1	cCOMOX_MSG_CODE_Hello message type code.

The sCOMOX_OUT_MSG_Hello message contains only the “Code” field and triggers a response from the iCOMOX of the sCOMOX_IN_MSG_Hello.

2.3. Reset Messages

2.3.1. sCOMOX_IN_MSG_Reset

```
typedef struct __attribute__((__packed__))
{
        eCOMOX_MSG_CODE         Code;
} sCOMOX_IN_MSG_Reset;
```

Field name	Field size (bytes)	Description
Code	1	cCOMOX_MSG_CODE_Reset message type code.

The sCOMOX_IN_MSG_Reset message is returned to the host after receiving the COMOX_OUT_MSG_Reset request message.

2.3.2. sCOMOX_OUT_MSG_Reset

```
typedef struct __attribute__((__packed__))
{
        eCOMOX_MSG_CODE         Code;
} sCOMOX_OUT_MSG_Reset;
```

Field name	Field size (bytes)	Description
Code	1	cCOMOX_MSG_CODE_Reset message type code.

The sCOMOX_OUT_MSG_Reset message requests the iCOMOX to reset itself.

Note: After reset, the iCOMOX returns to its default configuration.

2.4. Get Configuration Messages

2.4.1. sCOMOX_IN_MSG_GetConfiguration

```
typedef struct __attribute__((__packed__))
{
        eCOMOX_MSG_CODE         Code;
        sCOMOX_CONFIGURATION    Configuration;
} sCOMOX_IN_MSG_GetConfiguration;
```

Field name	Field size (bytes)	Description
Code	1	cCOMOX_MSG_CODE_GetConfiguration message type code.
Configuration	3	sCOMOX_CONFIGURATION structure.

The sCOMOX_IN_MSG_GetConfiguration message is returned to the host after receiving the sCOMOX_OUT_MSG_GetConfiguration request message. It returns the current configuration of the iCOMOX.

2.4.2. sCOMOX_OUT_MSG_GetConfiguration

```
typedef struct __attribute__((__packed__))
{
        eCOMOX_MSG_CODE         Code;
} sCOMOX_OUT_MSG_GetConfiguration;
```

Field name	Field size (bytes)	Description
Code	1	cCOMOX_MSG_CODE_GetConfiguration message type code.

The sCOMOX_OUT_MSG_GetConfiguration message requests the iCOMOX to return its current configuration.

2.5. Set Configuration Messages

2.5.1. sCOMOX_IN_MSG_SetConfiguration

```
typedef struct __attribute__((__packed__))
{
        eCOMOX_MSG_CODE         Code;
} sCOMOX_IN_MSG_SetConfiguration;
```

Field name	Field size (bytes)	Description
Code	1	cCOMOX_MSG_CODE_SetConfiguration message type code.

The sCOMOX_IN_MSG_SetConfiguration message acknowledges the receiving of the sCOMOX_OUT_MSG_SetConfiguration request message.

2.5.2. sCOMOX_OUT_MSG_SetConfiguration

```
typedef struct __attribute__((__packed__))
{
        eCOMOX_MSG_CODE             Code;
        sCOMOX_CONFIGURATION        Configuration;
} sCOMOX_OUT_MSG_SetConfiguration;
```

Field name	Field size (bytes)	Description
Code	1	cCOMOX_MSG_CODE_SetConfiguration message type code.
Configuration	3	sCOMOX_CONFIGURATION structure.

The sCOMOX_OUT_MSG_SetConfiguration message requests the iCOMOX to change its current configuration with the one that is provided in the “Configuration” field.

2.6. Report Message

2.6.1. sCOMOX_IN_MSG_Report

The sCOMOX_IN_MSG_Report is the central message in the iCOMOX system. Its purpose is to report the raw data that is sampled from the various sensors of the iCOMOX.

This is the only message that is unidirectional (from the iCOMOX to the host only; there is no explicit request message for it).

This message also has 5 different forms, depending on the kind of the sensor about which the message reports.

The message structure is displayed below:

```
typedef struct __attribute__((__packed__))
{
        eCOMOX_MSG_CODE                 Code;
        eCOMOX_SENSOR                   PayloadType;
        union __attribute__((__packed__))
        {
                uint8_t                 payload[0];
                sPAYLOAD_ADXL362        ADXL362;
                saPAYLOAD_ADXL356B      ADXL356B;
                s16aPAYLOAD_BMM150      BMM150;
                sPAYLOAD_ADT7410        ADT7410;
                sPAYLOAD_IM69D130       IM69D130;
        };
} sCOMOX_IN_MSG_ReportBuffer;
```

The sCOMOX_IN_MSG_Report constants and related defined types are displayed below:

```
typedef enum __attribute__((__packed__))
{
	cCOMOX_SENSOR_ADXL362,		// Low power accelerometer
	cCOMOX_SENSOR_ADXL356B,		// Accelerometer + temperature
	cCOMOX_SENSOR_BMM150,		// Magnetometer
	cCOMOX_SENSOR_ADT7410,		// Temperature sensor
	cCOMOX_SENSOR_IM69D130,		// Microphone

	cCOMOX_SENSOR_COUNT,
} eCOMOX_SENSOR;
#define IM69D130_UNCOMPRESSED_SAMPLES_NUM		(3000)

typedef struct __attribute__((__packed__))
{
	uADXL362_REGS  Registers;		// Read all the useful registers map of the ADXL362. May change to less registers in the future
} sPAYLOAD_ADXL362;

#define ADXL356B_SAMPLES_NUM_PER_PIN		(2000)

typedef struct __attribute__((__packed__))
{
	int16_t  AccX;
	int16_t  AccY;
	int16_t  AccZ;
	int16_t  Temp;
} saPAYLOAD_ADXL356B[ADXL356B_SAMPLES_NUM_PER_PIN];		// TBD: Maybe it will be possible to separate the temp, accX, accY, accZ to arrays instead of the interleaved format

#define BMM150_SAMPLES_NUM  (300)

typedef struct __attribute__((__packed__))
{
```

```
        int16_t         Bx;
        int16_t         By;
        int16_t         Bz;
        uint16_t R;
} s16aPAYLOAD_BMM150[BMM150_SAMPLES_NUM];

typedef struct __attribute__((__packed__))
{
        int16_t         Temp_Q9_7;
        //uADT7410_REGS         Registers;                                      // Read all the useful
registers map of the ADT7410. May change to less registers in the future
} sPAYLOAD_ADT7410;

#define IM69D130_PAYLOAD_SAMPLES_NUM        (1290)
typedef struct __attribute__((__packed__))
{
        int16_t  s16aPAYLOAD_IM69D130[IM69D130_PAYLOAD_SAMPLES_NUM];            // TBD:
microphone buffer sizes will be later determined
} sPAYLOAD_IM69D130;
```

Field name	Field size (bytes)	Description
Code	1	cCOMOX_MSG_CODE_Report message type code.
PayloadType	1	eCOMOX_SENSORenum type. The value in this field determines which sensor data is provided in the payload field (and thus determines the message's actual size).
	Depends on the value of the PayloadType field: ADXL362 – 47 ADXL356B – 16000 BMM150 –2400 ADT7410 – 2 IM69D130 - 2580	This is an anonymous field, which allows direct access to the sensor data structs. It can contain register map of the ADXL362 accelerator, acceleration data from the ADXL356B accelerator, magnetic fields from the BMM150 magnetometer, temperature from the ADT7410 and voice samples from the microphone IM69D130.

3. Configuration Structure

The configuration is stored in a 3 bytes struct as follows:

```
typedef struct __attribute__((__packed__))
{
	eCOMOX_SENSOR_BITMASK		Reports;
	eCOMOX_COMM_CHANNEL		CommChannel;
	union __attribute__((__packed__)) uCOMOX_CONFIGURATION_Activate
	{
		uint8_t		value;
		struct __attribute__((__packed__))
		{
			bool	LED1		: 1;	// Green LED
			bool	LED2		: 1;	// Red LED
			bool	Vibrator : 1;
		};
	} Activate;		// For each bit field: "1" to activate, "0" to de-activate
} sCOMOX_CONFIGURATION;
```

With the following type defs:

```
typedef enum __attribute__((__packed__))
{
	cCOMOX_SENSOR_BITMASK_ADXL362		= 1 << 0,
	cCOMOX_SENSOR_BITMASK_ADXL356B		= 1 << 1,
	cCOMOX_SENSOR_BITMASK_BMM150		= 1 << 2,
	cCOMOX_SENSOR_BITMASK_ADT7410		= 1 << 3,
	cCOMOX_SENSOR_BITMASK_IM69D130		= 1 << 4,

	cCOMOX_SENSOR_BITMASK_TOP		= 1 << 4,
} eCOMOX_SENSOR_BITMASK;

typedef enum __attribute__((__packed__))
{
	cCOMOX_COMM_CHANNEL_USB,
```

```
        cCOMOX_COMM_CHANNEL_SMIP,


        cCOMOX_COMM_CHANNEL_COUNT,
} eCOMOX_COMM_CHANNEL;
```

Field name	Field size (bytes)	Description
Reports	1	This is a bitmask of type eCOMOX_SENSOR_BITMASK. Each of its 5 LSB bits is referred to different sensor. A bit with a value of "1" means to send the sCOMOX_IN_MSG_Report for the corresponding sensor. A bit with value of "0" means to avoid doing it. To stop all the report message from the iCOMOX, set the value of this field to 0.
CommChannel	1	This is an enum type eCOMOX_COMM_CHANNEL. Its value determines the destination of the various sCOMOX_IN_MSG_Report messages. Currently, the options are to redirect these messages to the USB or to the SmartMesh. Note that the value of this field does not correspond to the destination of the response messages of the non report messages.
Activate	1	This is a union uCOMOX_CONFIGURATION_Activate, which includes a bitmask struct. Its purpose it to enable ("1") or disable ("0") the green LED, the red LED, and/or the vibrator.

The configuration struct contains the iCOMOX state. Affecting the iCOMOX normal operation is done implicitly by changing the iCOMOX state.

4. Connecting and Configuring the iCOMOX

To connect and configure the iCOMOX, perform the following steps:

1. When a valid sCOMOX_IN_MSG_Hello message is received, go to step **4.**
2. Send sCOMOX_OUT_MSG_Hello message.
3. Receive incoming messages for a short time and go to step **1.**
4. Send sCOMOX_OUT_MSG_SetConfiguration message with the desired kind of reports.
5. Receive incoming messages for a short time.
6. When a valid sCOMOX_IN_MSG_Report message is received, then it and go to step **5** again.
7. Go to step **4.**

5. Document Revision History

Revision	Date	Author	Status and Description
2.0	02.09.2019	Kobi de Trenewan	Initial version
2.0	04.09.2019	M Elias	Revision

T. +972.3.943.5050 F. +972.3.943.5055 E. info@shiratech-solutions.com

58 Amal St, Kiryat Arie POB 3272, PetachTikva 4951358, Israel

www.shiratech-solutions.com