
Getting started with the software package for STEVAL-IDI001V1, based on STM32Cube

Introduction

This document describes how to get started with the STEVAL-IDI001V1 software.

STEVAL-IDI001V1 software provides a complete framework for STM32 to build applications using heterogeneous sensor data. It is easily portable across different MCU families thanks to STM32Cube™. This package contains a sample application that acquires raw data from different kinds of sensors, packs it into a custom protocol and streams it via USB (Virtual COM Port). Moreover, the SW is able to stream real time audio data to a PC via a standard USB audio-input driver.

This software provides a sample implementation for the STEVAL-IDI001V1 board, which is equipped with an STM32F439 MCU, a motion sensor (IMU 9axes LSM9DS1), an environmental sensor for pressure (LPS25HB), humidity and temperature (HTS221), a UV index (UVIS25), proximity and ambient light sensors (VL6180x) and an omnidirectional digital microphone (MP34DT01).

The software is based on STM32Cube technology and expands the STM32Cube based range of packages.

Contents

1	What is STM32Cube?	4
1.1	STM32Cube architecture	4
2	STEVAL-IDI001V1 software, expansion for STM32Cube	6
2.1	Overview	6
2.2	Architecture	6
2.3	Folders structure	7
2.4	APIs	8
2.5	Sample application description.....	8
2.5.1	Application architecture	8
2.5.2	Sensors acquisition process.....	9
2.5.3	Microphones acquisition processes	9
2.6	STCmdP: ST command protocol.....	10
2.6.1	Data format.....	10
2.6.2	Process involved in sending/receiving packets	10
2.6.3	Checksum Algorithm	11
2.6.4	Byte stuffing.....	12
2.6.5	Standard commands	12
2.6.6	CMD_Ping Example	13
2.6.7	Commands available	14
3	PC Utility	16
3.1	Data logger example	16
3.2	Data logger demo.....	16
3.3	PC audio recording utility example: Audacity	17
4	System setup guide	19
4.1	Hardware description	19
4.1.1	STEVAL-IDI001V1.....	19
4.2	Software description.....	19
4.3	Hardware and Software setup.....	20
4.3.1	Hardware setup	20
4.3.2	Software setup.....	20
4.3.3	Data Logger demo setup Guide	21
4.3.4	SD card DataLog example	23
5	Acronyms and Abbreviations	25
6	References	26

7 **Revision history** **27**

1 What is STM32Cube?

STMCube™ represents the STMicroelectronics initiative to make developers' lives easier by reducing development effort, time and cost. STM32Cube covers the STM32 portfolio.

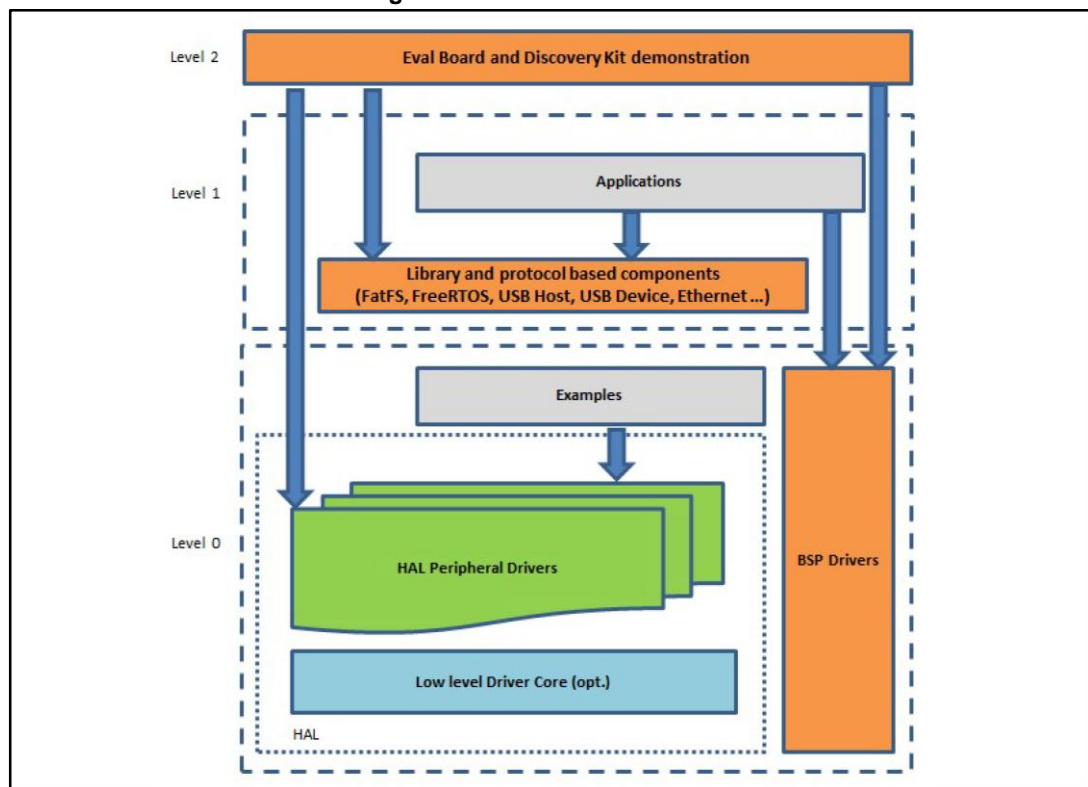
STM32Cube version 1.x includes:

- STM32CubeMX, a graphical software configuration tool that allows the generation of C initialization code using graphical wizards.
- A comprehensive embedded software platform specific to each series (such as the STM32CubeF4 for the STM32F4 series), which includes:
 - the STM32Cube HAL embedded abstraction-layer software, ensuring maximized portability across the STM32 portfolio
 - a consistent set of middleware components such as RTOS, USB, TCP/IP and graphics
 - all embedded software utilities with a full set of examples

1.1 STM32Cube architecture

The STM32Cube firmware solution is built around three independent levels that can easily interact with one another, as described in the diagram below:

Figure 1: Firmware architecture



Level 0: This level is divided into three sub-layers:

- Board Support Package (BSP): this layer offers a set of APIs relative to the hardware components in the hardware boards (Audio codec, IO expander, Touchscreen, SRAM driver, LCD drivers. etc...) and composed of two parts:

- Component: is the driver relative to the external device on the board and not related to the STM32, the component driver provides specific APIs to the external components of the BSP driver, and can be ported on any other board.
- BSP driver: links the component driver to a specific board and provides a set of easy to use APIs. The API naming convention is BSP_FUNCT_Action(): e.g., BSP_LED_Init(), BSP_LED_On().

It is based on modular architecture allowing it to be easily ported on any hardware by just implementing the low level routines.

- Hardware Abstraction Layer (HAL): this layer provides the low level drivers and the hardware interfacing methods to interact with the upper layers (application, libraries and stacks). It provides generic, multi-instance and function-oriented APIs to help offload user application development time by providing ready to use processes. For example, for the communication peripherals (I2S, UART, etc.) it provides APIs for peripheral initialization and configuration, data transfer management based on polling, interrupt or DMA processes, and communication error management. The HAL Drivers APIs are split in two categories: generic APIs providing common, generic functions to all the STM32 series and extension APIs which provide special, customized functions for a specific family or a specific part number.
- Basic peripheral usage examples: this layer houses the examples built around the STM32 peripherals using the HAL and BSP resources only.

Level 1: This level is divided into two sub-layers:

- Middleware components: set of libraries covering USB Host and Device Libraries, STemWin, FreeRTOS, FatFS, LwIP, and PolarSSL. Horizontal interaction among the components in this layer is performed directly by calling the feature APIs, while vertical interaction with low-level drivers is managed by specific callbacks and static macros implemented in the library system call interface. For example, FatFs implements the disk I/O driver to access a microSD drive or USB Mass Storage Class.
- Examples based on the middleware components: each middleware component comes with one or more examples (or applications) showing how to use it. Integration examples that use several middleware components are provided as well.

Level 2: This level is a single layer with a global, real-time and graphical demonstration based on the middleware service layer, the low level abstraction layer and basic peripheral usage applications for board-based functions.

2 STEVAL-IDI001V1 software, expansion for STM32Cube

2.1 Overview

This software package expands the functionality of the STM32Cube platform.

The key features of the package are:

- Data logger sample application.
- Data storage on SD card.
- Complete middleware to easily communicate with a client application using a proprietary protocol.
- Real time operating system (FreeRTOS) to implement multitasks applications.
- Audio+CDC class USB driver to allow the recognition of the device as a standard USB microphone and a Virtual COM Port.
- Easy portability across different MCU families thanks to STM32Cube.
- Free user-friendly license terms.

This software enables data acquisition from different kinds of sensors, such as motion sensors, environmental sensors, proximity and ambient light sensors via I²C or SPI for some of them. Moreover, the data from up to four digital onboard MEMS microphones or up to eight external MEMS microphones can be acquired using four GPIOs. The acquired data can then be converted from PDM to the PCM audio communication and processing standard.

Exploiting the capabilities of an included VCP and standard audio-input USB driver, the device is recognized as a multichannel USB microphone and as a Virtual COM Port by Microsoft Windows or any Unix-like system. The audio can be recorded using any standard audio recording software, while for all the other functions, the C++ source code of the proprietary protocol is provided inside the software package.

You can also save the data on a microSD card if there is one plugged into the relevant connector.

2.2 Architecture

This software is an expansion for STM32Cube, as such it fully complies with the STM32Cube architecture.

The software is based on the STM32CubeHAL, the hardware abstraction layer for the STM32 microcontroller. The package extends STM32Cube by providing a Board Support Package (BSP) for the STEVAL-IDI001V1 board and a component driver for each specific sensor; each driver supports multiple instances of the same sensor at the same time. Moreover the software package contains a middleware layer with components for serial communication, the Real-Time operating system FreeRTOS for multitasking application and the audio library (see [Section 6: "References"](#)) needed for the PDM to PCM conversion.

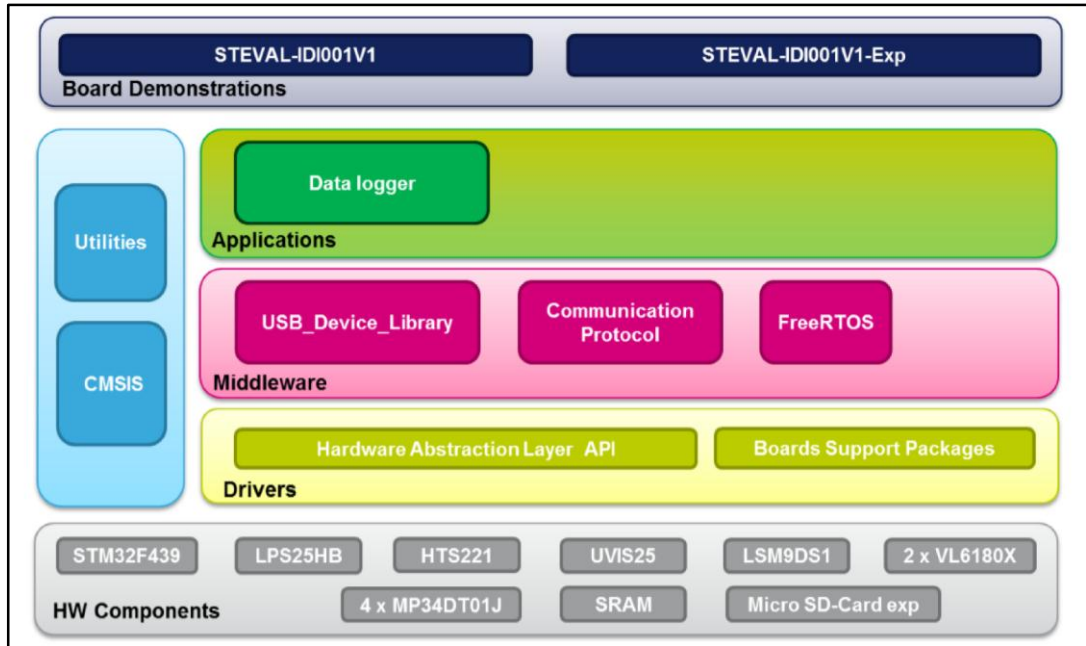
The software layers used by the application software to access and use the STEVAL-IDI001V1 board are the following:

- **STM32Cube HAL layer:** consists of a set of simple, generic, multi-instance APIs (application programming interfaces) which interact with the upper layer applications, libraries and stacks. These generic and extension APIs are based on a common framework which allows any layers they built on, such as the middleware layer, to implement their functions without requiring specific hardware information for a given

microcontroller unit (MCU). This structure improves library code reusability and guarantees easy portability across other devices.

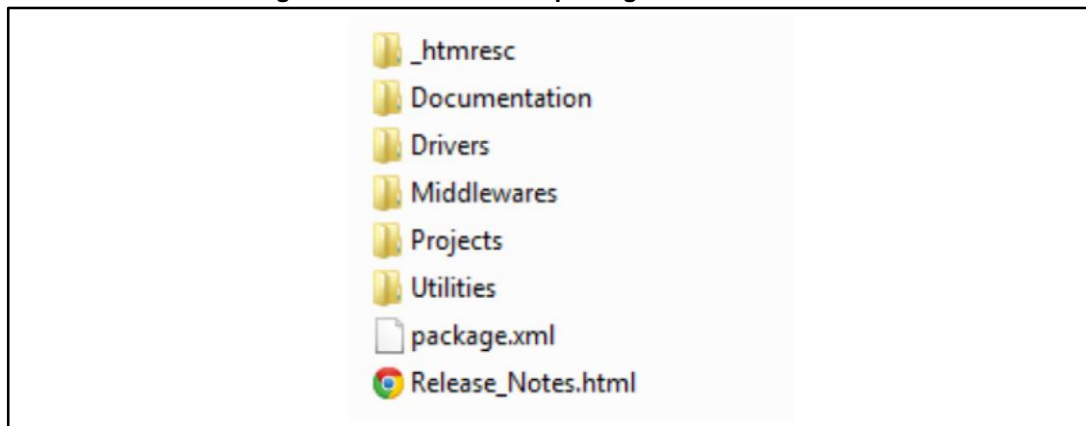
- **Board Support Package (BSP) layer:** provides software support for the STM32 Nucleo board peripherals, excluding the MCU. These specific APIs provide a programming interface for certain board specific peripherals like LEDs, user buttons, etc and can also be used to fetch individual board version information. It also provides support for initializing, configuring and reading data.

Figure 2: STEVAL-IDI001V1 software architecture



2.3 Folders structure

Figure 3: X-CUBE-MEMS1 package folder structure



The following folders are included in the software package:

- **Documentation:** contains a compiled HTML file generated from the source code with software component and API details.
- **Drivers:** contains the HAL drivers, the board specific drivers for each supported board or hardware platform, including the onboard components and the CMSIS vendor-independent hardware abstraction layer for the Cortex-M processor series.

- **Middleware:** contains libraries and protocols for the PDM to PCM conversion process, the audio-input USB driver and the real-time operating system FreeRTOS.
- **Projects:** contains a sample application to initialize and configure all the sensors, acquire data and send it in an encapsulated proprietary protocol via USB or store it on a microSD card. This application is available in three development environments (IAR Embedded Workbench for ARM, RealView Microcontroller Development Kit (MDK-ARM), Atollic TrueSTUDIO® for ARM).
- **Utilities:** contains the extra content needed for system setup, such as the PC software utility and demonstration software.

2.4 APIs

Detailed technical information regarding the user APIs can be found in a compiled HTML file inside the package Documentation folder, with full function and parameter descriptions.

2.5 Sample application description

An example application using the STEVAL-IDI001V1 is provided in the "Projects" directory. Ready to use projects are available for multiple IDEs.

2.5.1 Application architecture

Following HAL initialization, SystemClock and RTC configuration, you need to configure the USB CDC-Audio interface:

- Initialize the USB descriptor using `USB_D_AUDIO_CDC_Init_Microphone_Descriptor(...)` based on the desired sampling frequency and channel number.
- Initialize the USB core and start the USB functions by calling: `USB_Init(...)`, `USB_RegisterClass(...)`, `USB_D_AUDIO_CDC_RegisterInterface(...)`, `USB_Start(...)`.

This application takes advantage of FreeRTOS in order to create multiple tasks that run concurrently. Service and app tasks, together with related semaphores, are created in the main function.

Service tasks:

- **SERIAL_THREAD:** manages the communication with an external device. The task is always active and waiting to receive a message. This message is encapsulated in a proprietary protocol named STCmdP, which is explained elsewhere in this document. Finally, the received message is handled by the `STCmdP_interpreter.c` file and a response is sent back if necessary.
- **INIT_THREAD:** manages start and stop commands for the datalog application. It works in two modes:
 - Autostart mode: if enabled (`AUTOSTART define = 1`), all the sensors are initialized and the data stream starts immediately.
 - Normal mode: if enabled (`AUTOSTART define = 0`), a start command is required to set up the datalog application.

App task:

- **READ_STREAM_THREAD:** reads the exact timestamp, thanks to the RTC peripherals, acquires data from all the sensors, encapsulates them in a STCmdP message and sends it through the selected interfaces. The data streaming period is set by the start command and a precise timer is initialized in the `INIT_THREAD`, every time the timer counter reaches the request period, an interrupt is generated and the

osStremSemaphore is released (see the callback in application_manager.c file). This triggers sensor data reading and streaming.

- SD_CARD_THREAD: this thread is activated when a microSD card is plugged into the relevant connector, and the user button is pressed. A dedicated semaphore is released every second; the sensor data is acquired and saved in a file.
- PDM_PCM_THREAD: this thread is activated whenever 1 ms of PDM audio data has been acquired. Then an audio process function converts the acquired data into PCM samples.
- DB_NOISE_THREAD: measures ambient noise; dBNoise_callback is called every 64 ms.

2.5.2 Sensors acquisition process

This application is configured by default in normal mode (AUTOSTART define = 0), which means that an initial command is needed to activate each sensor. When the corresponding command is received, the sensor is initialized; if the procedure ends correctly, an ACK message is sent back to the host, otherwise, a NACK message is sent.

If the application is configured in Autostart mode (AUTOSTART define = 1), all the sensors are initialized automatically during the startup phase.

When the datalog application is started, the microcontroller reads the data from the active sensors via I²C or SPI. Dates are then included in an STCmdP message and sent via USB.

All the APIs needed to interact with the sensors are implemented in the BSP layer inside a file named steval_idi001v1_<sensortype>.c. The link functions necessary to communicate with the sensors are instead located in steval_idi001v1.c.

2.5.3 Microphones acquisition processes

A digital MEMS microphone can be acquired through peripherals like SPI, I²S or GPIO. It requires an input clock, and outputs a PDM stream of the same. This PDM stream is further filtered and decimated for conversion into the PCM standard format for audio transmission.

In this scenario, microphone acquisition works in the following way:

- a precise clock signal is generated by a timer and provided to the microphones; the DMA operates at a same frequency.
- A software demuxing step separates the signal from the two microphones and allows further processing like PDM to PCM conversion.

You will find additional resource information regarding MEMS microphones and PDM to PCM decimation in [Section 6: "References"](#).

By configuring analog switch (U18) through the firmware, you can acquire four external microphones on the same GPIOs instead of the four available on the board. You can also acquire additional external microphones, up to a maximum of eight, through the J9 connector.

In the firmware, audio-related components are collected in the application_audio.c file, which uses the dedicated BSP layer steval_idi001v1_audio_in.c and the PDM to PCM decimation library middleware.

To set the system up for four-microphone acquisition and two-channel streaming requires the following steps, which you can trace in the application_audio.c file:

- Initialize acquisition peripherals: the function Init_Acquisition_Peripherals(...) initializes the PDM to PCM middleware and sets up the required MCU peripherals with the dedicated BSP function BSP_AUDIO_IN_Init(...).
- start acquisition using BSP_AUDIO_IN_Record(...).

The audio streaming via USB is always active; you can use software to record and edit sounds.

2.6 STCmdP: ST command protocol

The section describes the bit-level message format and the higher level commands of the communication protocol used in this application.

2.6.1 Data format

The serial protocol defines two layers:

- Layer 1 - the first lower layer
- Layer 2 - a second upper layer

The Layer 1 packet is defined as follows:

Table 1: Layer 1 packet

Type	Encoded Payload	EOF
Bytes	N	1

Where:

- **Encoded Payload:** is the message exchanged by the protocol between the module and a host. Its length can change but it is limited by the physical MTU (Maximum Transmission Unit). The actual payload is encoded by a byte stuffing function described elsewhere in this document, to avoid any EOF characters in the content.
- **EOF (0xF0):** byte representing the end of the packet. It has the value: 0xF0.

The Encoded Payload of Layer 1 must be processed by a function which performs reverseByteStuffing, described elsewhere in this document, to obtain the Payload. The resulting Payload is a Layer 2 packet with the following format:

Table 2: Layer 2 packet

Type	Destination Address	Source Address	Command CMD	Payload	CHK
Bytes	1	1	1	N	1

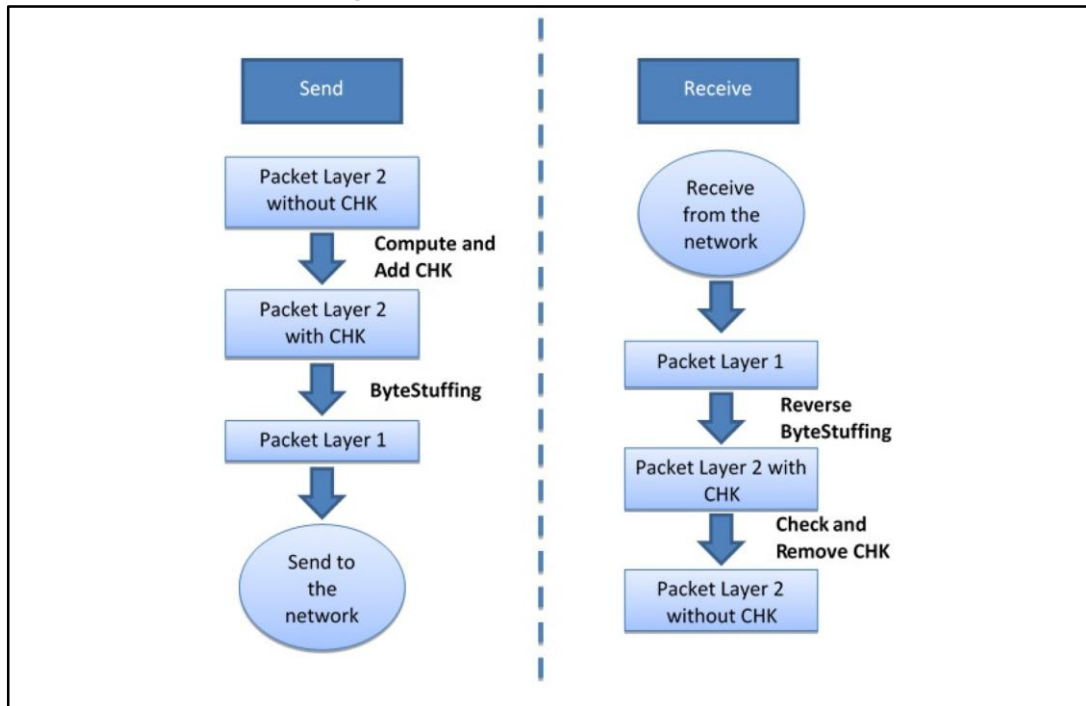
Where:

- **Destination Address:** symbolic address of the destination. It enables unicast messages on a shared bus.
- **Source Address:** symbolic address of the source.
- **Command:** code representing the issued command. It specifies how the payload should be interpreted.
- **Payload:** data which is interpreted with respect to the CMD field.
- **CHK:** the computed sum of all the bytes in the packet must be zero.

2.6.2 Process involved in sending/receiving packets

The following figure shows the packet send and receive processes. The two checksum and byte stuffing functions and their respective reverse functions, ensure correct CMD and corresponding payload interpretation.

Figure 4: Send and receive processes



2.6.3 Checksum Algorithm

The checksum algorithm ensures that the packet you handle contains the correct information.

The algorithms needed to compute and to verify the checksum integrity are explained by the following pseudo-code snippets.

ComputeandAddCHK

```

begin
uint_8 CHK = 0
for i=0:layer2length
CHK = CHK - layer2data(i)
end
layer2data(layer2length+1) = CHK
end
  
```

CheckandRemoveCHK

```

begin
uint_8 CHK = 0
for i=0:layer2length
CHK = CHK + layer2length
end
layer2length = layer2length - 1
  
```

```
return CHK == 0
end
```

2.6.4 Byte stuffing

Byte stuffing is a process that transforms a sequence of data bytes that may contain 'illegal' or 'reserved' values into a potentially longer sequence with none of these values.

In our case, as the EOF character identifies the end of the packet, there must be no other occurrence of this character in the packet.

For this reason the following special characters are defined:

- `TMsg_EOF (0xF0)`: is the EOF of layer 1 packet
- `TMsg_BS (0xF1)`: is the byte stuffing escape character
- `TMsg_BS_EOF (0xF2)`: is the substitution for `TMsg_EOF`

The byte stuffing algorithm used in sending actions is defined as follows:

- Given a Layer 2 message, for each character:
 - Substitute `TMsg_BS` with `TMsg_BS` followed by `TMsg_BS` (double the character). `0xF1` becomes `0xF1 0xF1`.
 - Substitute each `TMsg_EOF` with `TMsg_BS` followed by `TMsg_BS_EOF`. `0xF0` becomes `0xF1 0xF2`.

As can be seen, the Layer 2 packet length may change in the process.

The reverse byte stuffing algorithm can be defined in the same way:

- Given a Layer 1 payload, for each character
 - Substitute the expression (`TMsg_BS` followed by `TMsg_BS`) with a single `TMsg_BS`. `0xF1 0xF1` becomes `0xF1`.
 - Substitute the expression (`TMsg_BS` followed by `TMsg_BS_EOF`) with `TMsg_EOF`. `0xF1 0xF2` becomes `0xF0`.

2.6.5 Standard commands

A set of fundamental and standard commands are defined in this document. The protocol can be extended in order to accommodate many more commands which are specifically related to the target application.

The standard commands are summarized in the following table:

Table 3: Standard commands

Command	CMD value	Meaning
CMD_Ping	0x01	This is the standard ping command, the device will reply accordingly
CMD_Read_PressString	0x02	Requests the presentation string which contains basic device information (name and version)
CMD_Reset	0x0F	Requests a reboot of the device
CMD_Reply_Add	0x80	This value is added to the value of the CMD field to form the command for the reply. E.g., 0x81 = reply to the PING command
CMD_NACK	0x03	The requested command procedure did not end correctly.

This list specifies the value of the CMD field which can be inserted in the request packet (from the host to the module). The module will reply by adding CMD_Reply_Add to the value of CMD.

Note that the length of the packets is determined by a low level function using terminator and escape special characters. For this reason, the real length of the packet may not equal the message length. Moreover, the data contained in the Layer 2 payload is serialized before being copied into a packet so that, using a function to deserialize them, the data is independent of the architecture (big/little-endian).

The commands in Layer 2 format are explained here in more detail:

CMD_Ping: The packet is formed as shown below

Table 4: CMD_Ping

Type	Destination address	Source address	Command
Length (Bytes)	1	1	1
Value	XX	YY	CMD_Ping

The module will answer with the same packet format but the source and destination addresses are obviously swapped and the command field will contain CMD_PING + CMD_Reply_Add.

CMD_Read_PresString: The request packet is equal to CMD_PING where CMD is substituted by CMD_Read_PresString.

The reply you will receive is:

Table 5: CMD_Reset

Type	Destination address	Source address	Command	Payload
Length (Bytes)	1	1	1	K
Value	YY	XX	CMD_Read_PresString + CMD_Reply_add	String of K characters

CMD_Reset: the request packet is equal to CMD_PING where CMD is substituted by CMD_Reset. The module will reboot and no ACK will be sent. The host can check the new state of the module using other application-dependent commands.

2.6.6 CMD_Ping Example

In this example, we consider the PING command, how to form the request and its response at "Layer 1" and "Layer 2" (see the general Serial Protocol document for more information about layers).

Request

For the following hypothetical values:

- Sender Address = TMsg_EOF = 0xF0
- Destination Address = 0x42

The Layer 2 request packet is:

Table 6: Layer 2 packet

Type	Destination address	Source address	Command	CHK
Length (bytes)	1	1	1	1
Value	0x42	0xF0	CMD_Ping = 0x01	205 = 0xCD

After the bytestuffing algorithm, the "Layer 1" packet is:

0x42	TMsg_BS = 0xF1	TMsg_BS_EOF = 0xF2	0x01	0xCD	TMsg_EOF = 0xF0
------	----------------	--------------------	------	------	-----------------

Reply

The packet is received at "Layer 1". After reverse bytestuffing and checksum, the original packet is parsed (at "Layer 2").

Upon a PING request, "Layer 2" will reply:

Table 7: Layer 1 packet

Type	Destination address	Source address	Command	CHK
Length (bytes)	1	1	1	1
Value	0xF0	0x42	CMD_PING + CMD_Reply_Add = 0x81	77 = 0x4D

After the bytestuffing algorithm, the "Layer 1" packet is:

TMsg_BS = 0xF1	TMsg_BS_EOF = 0xF2	0x42	0x81	0x4D	TMsg_EOF = 0xF0
----------------	--------------------	------	------	------	-----------------

2.6.7 Commands available

The following table gives all the available commands used in the firmware example.

Table 8: Commands firmware

Type	Command	Code
Generic	CMD_Ping	0x01
	CMD_Read_PresString	0x02
	CMD_NACK	0x03
	CMD_Start_Data_Streaming	0x08
	CMD_Stop_Data_Streaming	0x09
	CMD_Set_DateTime	0x0C
	CMD_Get_DateTime	0x0D
	CMD_Reset	0x0F
	CMD_Reply_Add	0x80
Audio	CMD_AudioModule_GetEnergydB	0x43
Proximity & Ambient light	CMD_BBx_Init	0x50
	CMD_BB_ReadData	0x52
Environmental	CMD_LPS25H_Init	0x60

Type	Command	Code
	CMD_LPS25H_Read	0x61
	CMD-HTS221_Init	0x62
	CMD-HTS221_Read	0x63
	CMD_UVIS3_Init	0x64
	CMD_UVIS3_Read	0x65
Inertial	CMD_LSM9DS1_Init	0x70
	CMD_LSM9DS1_9AXES_Read	0x71
	CMD_LSM9DS1_ACC_Read	0x73
	CMD_LSM9DS1_GYR_Read	0x74
	CMD_LSM9DS1_MAG_Read	0x75

3 PC Utility

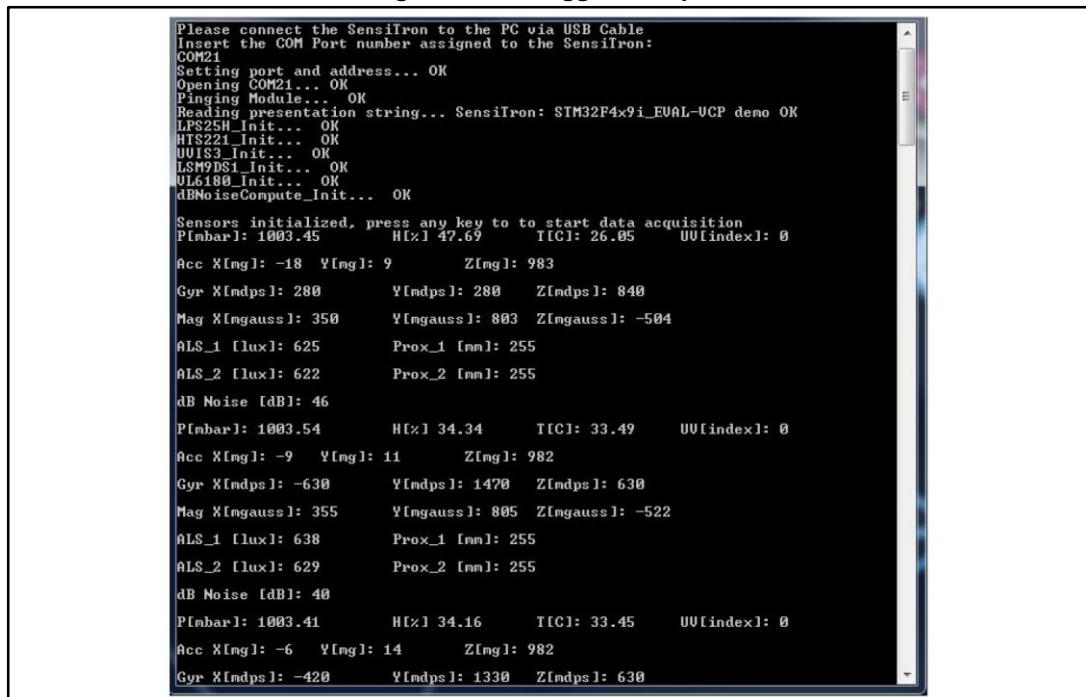
3.1 Data logger example

The STEVAL-IDI001V1 software package includes a simple Qt project that shows how to create a communication link between your PC and STEVAL-IDI001V1. This example is provided in the source code and contains a library with all the available commands to interact with the STEVA-IDI001V1.

The operational sequence is:

1. the Virtual COM Port indicated by the user is opened
2. an initialization message is sent to the STEVA-IDI001V1 for each sensor
3. raw data is acquired in a loop that runs until a button is pressed by the user.

Figure 5: Data logger example



```

Please connect the SensiTron to the PC via USB Cable
Insert the COM Port number assigned to the SensiTron:
COM21
Setting port and address... OK
Opening COM21... OK
Pinging Module... OK
Reading presentation string... SensiTron: STM32F4x9i_EVAL-UCP demo OK
LPS25H_Init... OK
HTS221_Init... OK
UUI33_Init... OK
LSM9DS1_Init... OK
UL6188_Init... OK
dBNoiseCompute_Init... OK

Sensors initialized, press any key to to start data acquisition
P[Inbar]: 1003.45      H[α]: 47.69      TIC]: 26.05      UU[index]: 0
Acc X[In]: -18      Y[In]: 9          Z[In]: 983
Gyr X[In]: 280      Y[In]: 280      Z[In]: 840
Mag X[In]: 350      Y[In]: 803      Z[In]: -504
ALS_1 [lux]: 625      Prox_1 [mm]: 255
ALS_2 [lux]: 622      Prox_2 [mm]: 255
dB Noise [dB]: 46
P[Inbar]: 1003.54      H[α]: 34.34      TIC]: 33.49      UU[index]: 0
Acc X[In]: -9      Y[In]: 11         Z[In]: 982
Gyr X[In]: -630     Y[In]: 1470     Z[In]: 630
Mag X[In]: 355      Y[In]: 805      Z[In]: -522
ALS_1 [lux]: 638      Prox_1 [mm]: 255
ALS_2 [lux]: 629      Prox_2 [mm]: 255
dB Noise [dB]: 40
P[Inbar]: 1003.41      H[α]: 34.16      TIC]: 33.45      UU[index]: 0
Acc X[In]: -6      Y[In]: 14         Z[In]: 982
Gyr X[In]: -420     Y[In]: 1330     Z[In]: 630
  
```

3.2 Data logger demo

A demonstration program (binary only) is provided with the STEVAL-IDI001V1 package.

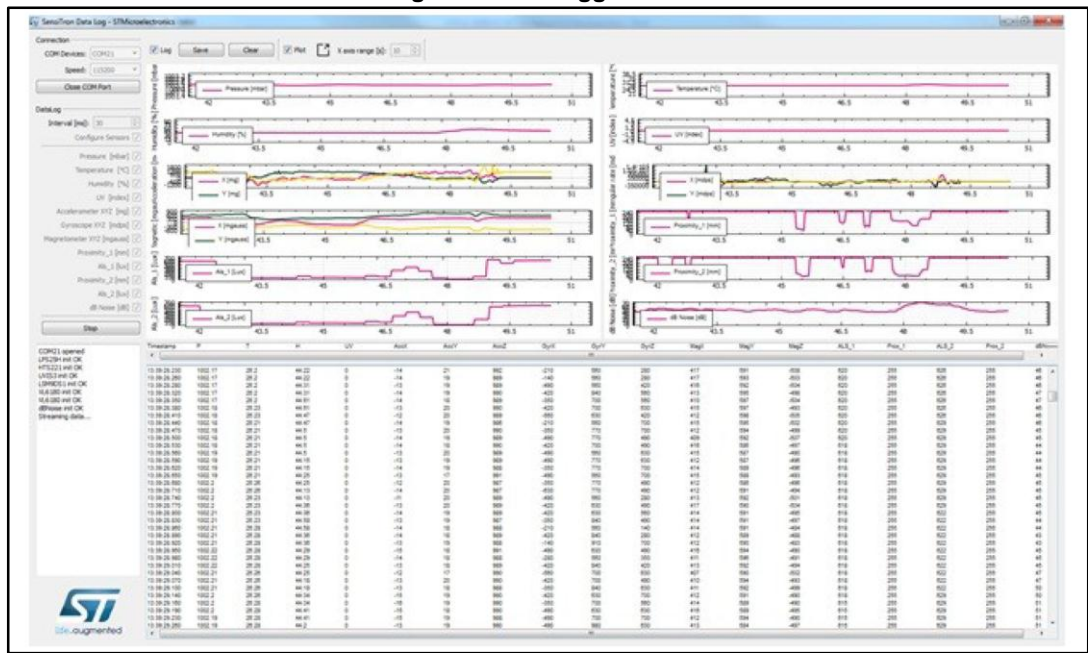
The aim of this software is to give an overview of all the sensors mounted on board.

Two different views are available:

- a graphics view that shows a plot for each measured quantity
- a log view that displays the acquired data with the corresponding timestamp

The incoming data can be saved in a text file in order to use it for sensor fusion algorithm development. A screenshot of the application is shown in [Figure 6: "Data logger demo"](#).

Figure 6: Data logger demo



3.3 PC audio recording utility example: Audacity

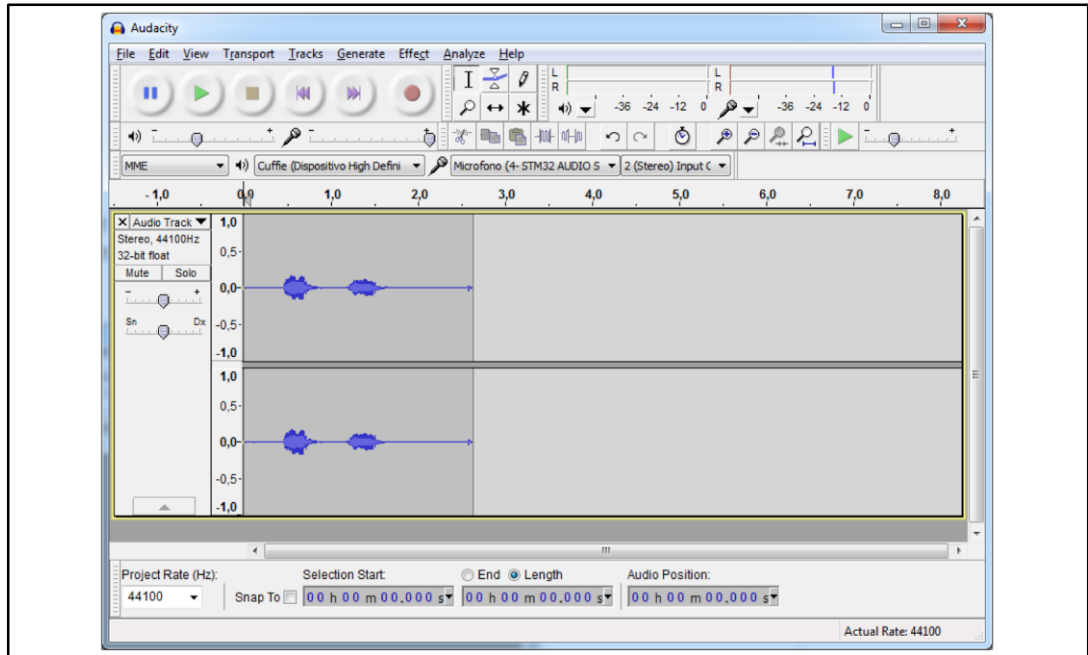
This section describes the use of the Audacity® free, open source, cross-platform software application for multiple channel audio recording and editing.

It is available for Windows®, Mac®, GNU/Linux® and other operating systems as a freeware audio editing environment (<http://audacity.sourceforge.net/?lang=en>).

In windows 7, the released Audacity version is capable of recording up to 2 microphones; the proprietary ASIO driver is usually the best way of making multi-channel recordings on Windows, but Licensing restrictions prevent including ASIO support in released versions of Audacity. However, it can be compiled with ASIO support for private, non-distributable use. For additional information resources, please refer to [Section 6: "References"](#).

To start audio recording, first check the audio input device is STM32 AUDIO Streaming in FS mode and then begin audio recording and subsequent playback using the respective buttons.

Figure 7: Audacity for windows



4 System setup guide

4.1 Hardware description

This section describes the hardware components needed for developing sensor-based applications.

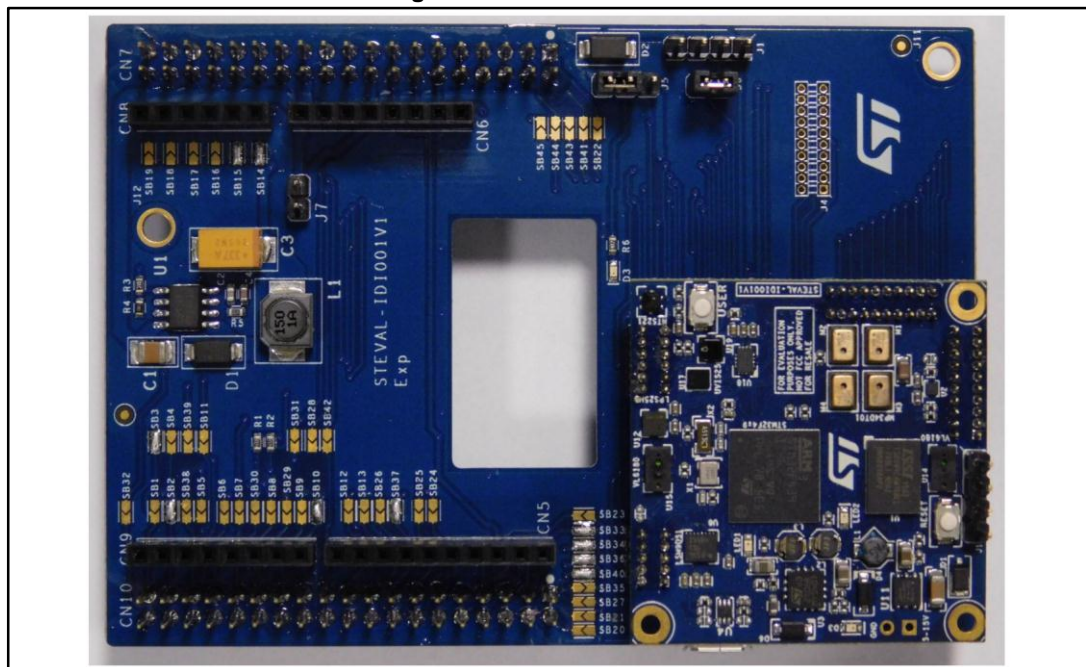
4.1.1 STEVAL-IDI001V1

STEVAL-IDI001V1 is an evaluation board based on STM32F4 Microcontroller and a wide range of ST sensors. With a small, 4x4 cm form factor, it represents an exemplary integration of sensors and CPU for "sensor-hub like" applications.

The hardware is compatible with the STM32-Nucleo and X-NUCLEO expansion board, thanks to the STEVAL-IDI001-Exp adapter.

The STEVAL-IDI001V1 board requires a separate probe in order to be programmed. Either an ST-LINK/V2-1 debugger/programmer or a Nucleo that integrates it can be used with an SWD cable adapter.

Figure 8: STEVAL-IDI001V1



Information about the STEVAL-IDI001V1 is available on www.st.com.

4.2 Software description

The following software components are required to setup the suitable development environment for creating applications for the STEVAL-IDI001V1:

- Firmware example: an expansion for STM32Cube dedicated to sensor application development. The STEVAL-IDI001V1 firmware and related documentation is available on st.com.
- ST-LINK/V2 USB driver available at ST-LINK driver.
- STM32 Virtual COM Port driver available at VCP driver (Optional).
- Development tool-chain and Compiler: The STM32Cube expansion software supports the three following environments:

- IAR Embedded Workbench for ARM® (EWARM) toolchain + ST-Link
- RealView Microcontroller Development Kit (MDK-ARM) toolchain + ST-LINK
- Atollic TrueSTUDIO® for ARM® Pro + ST-LINK

4.3 Hardware and Software setup

This section describes the hardware and software setup procedures.

4.3.1 Hardware setup

The following hardware components are needed:

- One STEVAL-IDI001V1.
- One USB type A to Micro-B USB cable for power supply and to connect the STEVAL-IDI001V1 to the PC.

4.3.2 Software setup

This section lists the minimum requirements to setup the SDK, run the sample testing scenario based on the previous description and customize your applications.

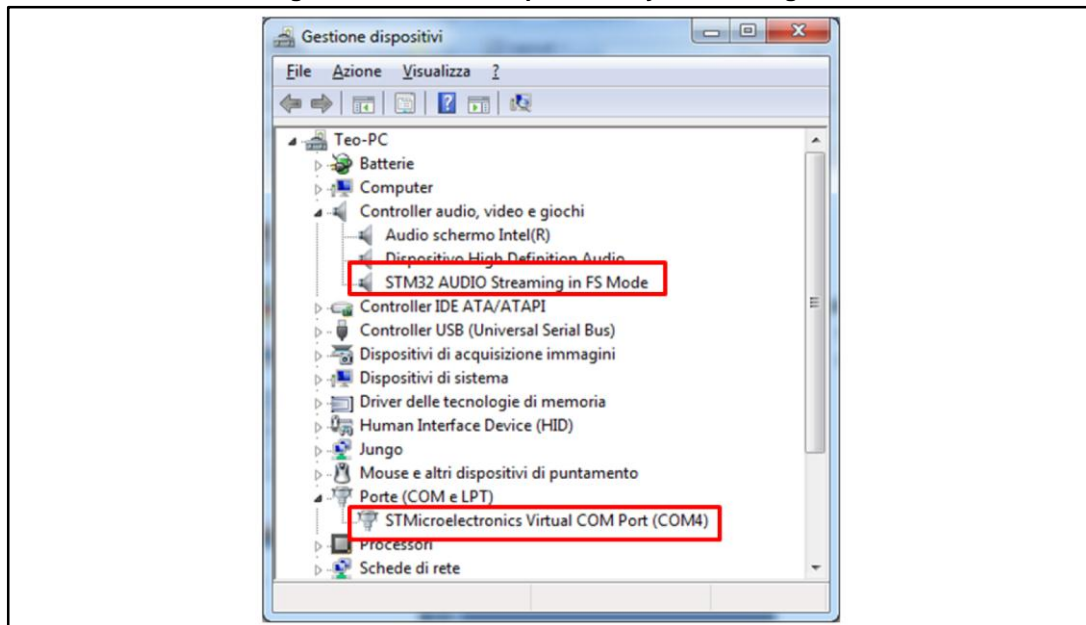
Development Tool-chains and Compilers

Choose one of the Integrated Development Environments supported by the STM32Cube expansion software and follow the system and setup details provided by the selected IDE provider.

Recognition of the device as a Virtual COM Port and as a standard USB microphone

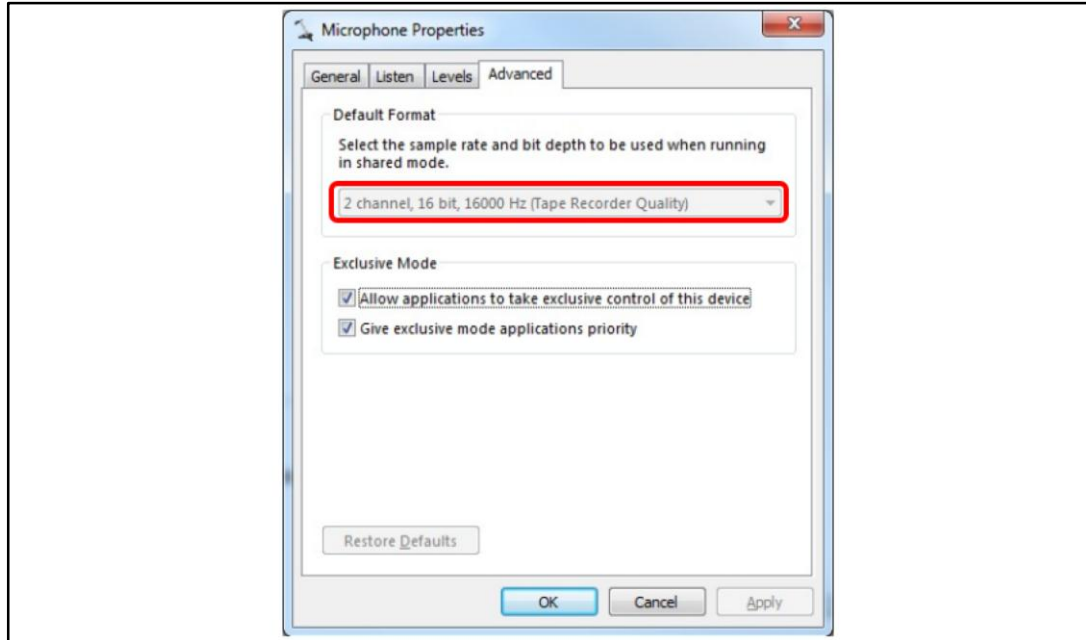
The software package includes a composite Audio+VCP driver that allows the device be recognized as a Virtual COM Port and as a standard USB microphone. Following firmware download to MCU FLASH and connection of the STEVAL-IDI001V1 board to your PC via USB cable, check the system manager to see if it has been recognized correctly, as depicted in [Figure 9: "STM32 microphone in system manager"](#).

Figure 9: STM32 microphone in system manager



Finally, right click on the volume icon on the windows task bar (in the bottom right part of the screen) and choose "recording device". Now select STM32 microphone and click on "Properties". In the "advanced" tab there is a summary of the current device setup in terms of sampling frequency and number of channels. Here, you should adjust the settings so you can record and save audio ([Figure 10: "Advanced configuration setup"](#)).

Figure 10: Advanced configuration setup

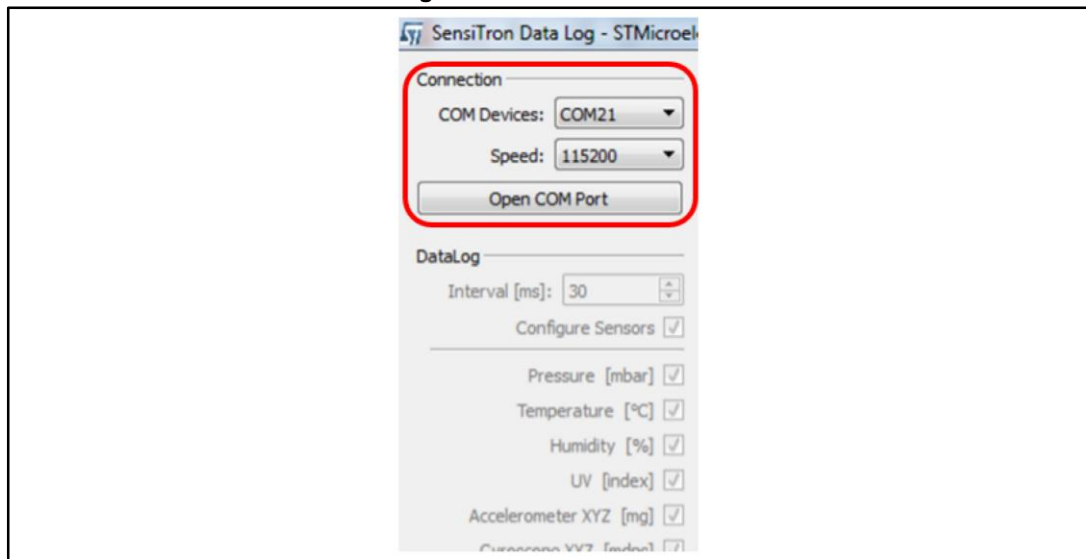


4.3.3 Data Logger demo setup Guide

Follow these steps to set up the Data Logger demo provided with the STEVAL-IDI001V1 software package:

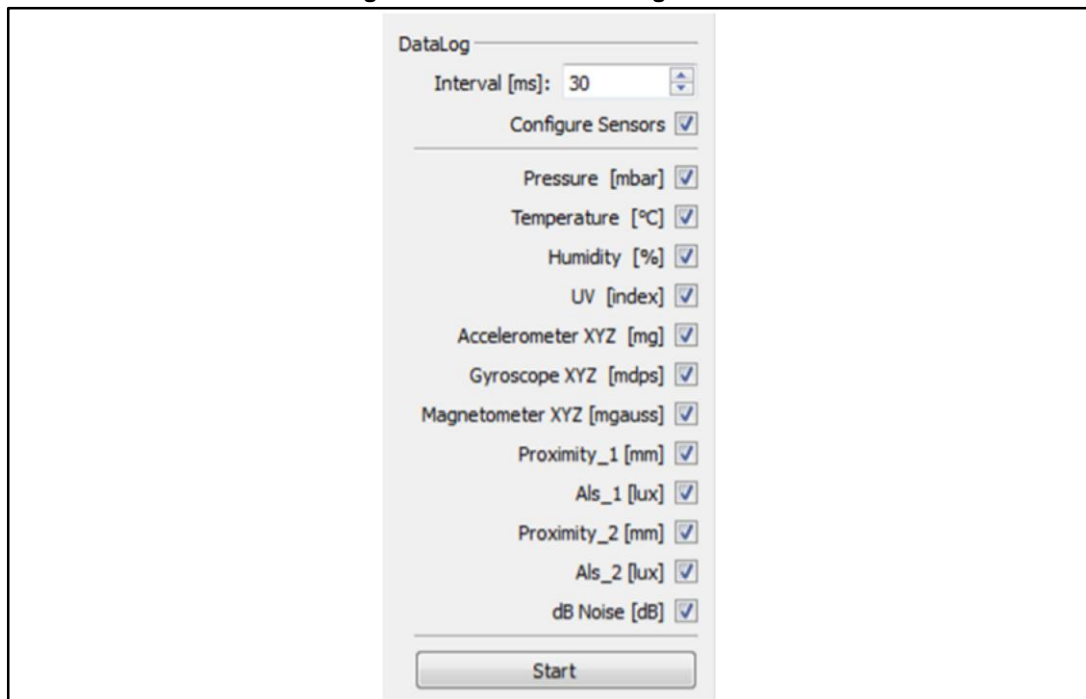
- Connect the STEVA-IDI001V1 board to the PC via USB.
- Launch the STEVAL-IDI001V1_Datalog.exe executable file; the COM ports of available ST devices will be listed as shown in [Figure 11: "SW connection"](#), select the correct one and click "Open COM Port".

Figure 11: SW connection



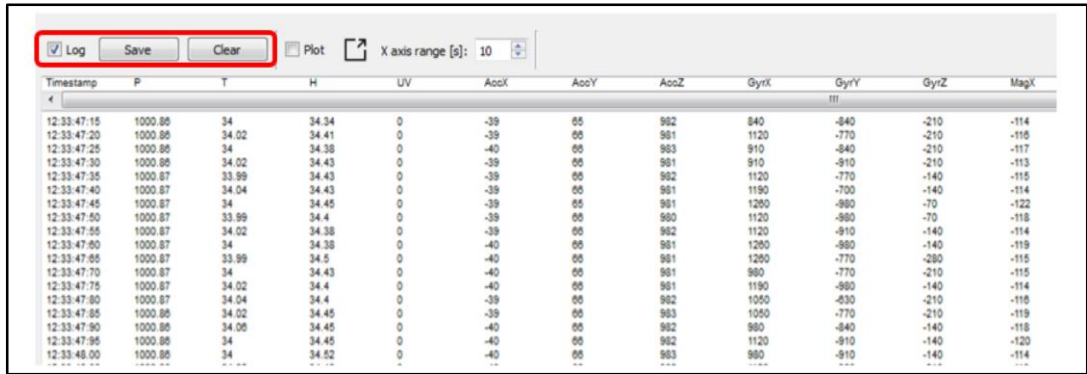
- Set the sensor sampling interval (in ms) and choose the sensors that you want to read data from. Press "Start" to commence data acquisition. ([Figure 12: "SW sensor configuration"](#)).

Figure 12: SW sensor configuration



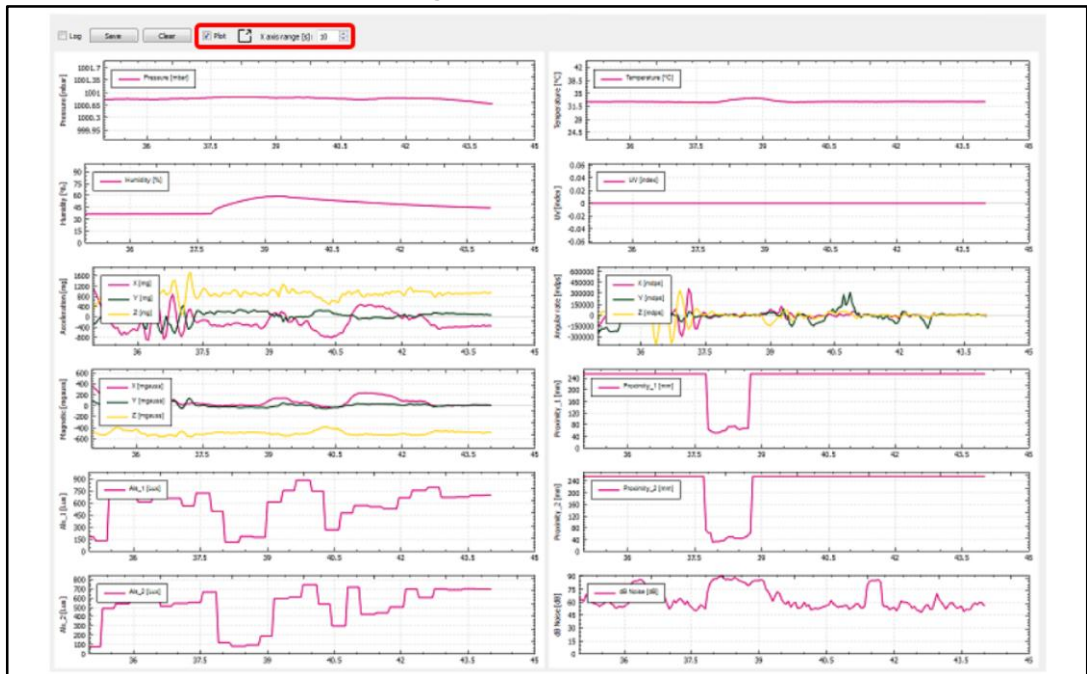
- Select the "Log" checkbox to display the data log; save the acquired data in a text file or clear the log content by pressing the corresponding "Save" and "Clear" buttons. ([Figure 13: "SW log view"](#)).

Figure 13: SW log view



- Select the "Plot" checkbox to display a graph for each selected sensor. You can attach and detach all the graphs to and from the main window and select the x-axis range (Figure 14: "SW plot view").

Figure 14: SW plot view



- Finally press the "Stop" button to arrest data acquisition, and "Close COM Port" button close the Virtual COM Port.

4.3.4 SD card DataLog example

The STEVAL-IDI001V1 software package also includes an example of how to use a microSD card for data storage. The firmware architecture of this example is similar to the sample data log application previously described in [Section 2.5: "Sample application description"](#).

In order to run this demo, insert the microSD card in the relevant connector on the STEVAL-IDI001V1. When you power the board, it automatically detects the microSD card and the red LED starts blinking.

Now press the user button to start the data log application (the red LED should turn off and the green one should start blinking). Every second, the timestamp and a value for each sensor are saved in a file called STEVAL_IDI001V1_Datalog_Nx.tsv where x is an incremental ID.

Press the user button again to stop the application.

You can restart the microSD card DataLog application any time you want, a new file with a different ID will be saved on the microSD card.

5 Acronyms and Abbreviations

Table 9: List of acronyms

Acronym	Description
PDM	Pulse Density Modulation
PCM	Pulse Code Modulation
USB	Universal Serial Bus
SPI	Serial Peripheral Interface
I ² S	Integrated Interchip Sound
I ² C	Inter-Integrated Circuit
BSP	Board Support Package
HAL	Hardware Abstraction Layer
IDE	Integrated Development Environment
STCmdP	ST Command Protocol

6 References

1. LSM9DS1 MEMS IMU 9axes inertial module, data. LSM9DS1 datasheet
2. STM32F439II High Performances MCU ARM®Cortex™-M4F. STM32F43911
3. LPS25HB 260-1260 hPa digital output barometer, data sheet. LPS25HB datasheet
4. HTS221 Capacitive digital sensor for relative humidity and temperature, datasheet. HTS221 datasheet
5. UVIS25 Digital UV Index, data sheet. UVIS25 datasheet
6. VL6180x Proximity and ambient light module, data sheet. VL6180x datasheet
7. MP34DT01 MEMS audio sensor omnidirectional digital microphone, data sheet. MP34DT01 datasheet
8. PDM audio software decoding on STM32 microcontrollers, Application note AN3998. PDM Application Note
9. Audacity ASIO Audio interface wiki. Audacity

7 Revision history

Table 10: Document revision history

Date	Revision	Changes
24-Nov-2015	1	Initial release.

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2015 STMicroelectronics – All rights reserved