Datasheet - Programming & Sensors

Measurement of range of motion during the night

Required materials, wiring and using a Micro:bit board

To program a micro:bit board to collect humidity and temperature, you will need the following hardware:

- Micro:bit V2 board and its integrated sensors: The main programmable board including an integrated light sensor via its LED display, an integrated sound level sensor and an integrated temperature sensor - Approximately 19 EUR per micro:bit
- Micro-USB cable: To power and program the micro:bit
- External battery (optional): For portable operation if the micro:bit needs to be detached - You will find the official micro:bit battery case available for purchase for around 2.20 EUR per pack

You can also buy the Micro:bit V2 kit including the USB cable and battery box for 21 EUR per kit or 177 EUR for 10 kits

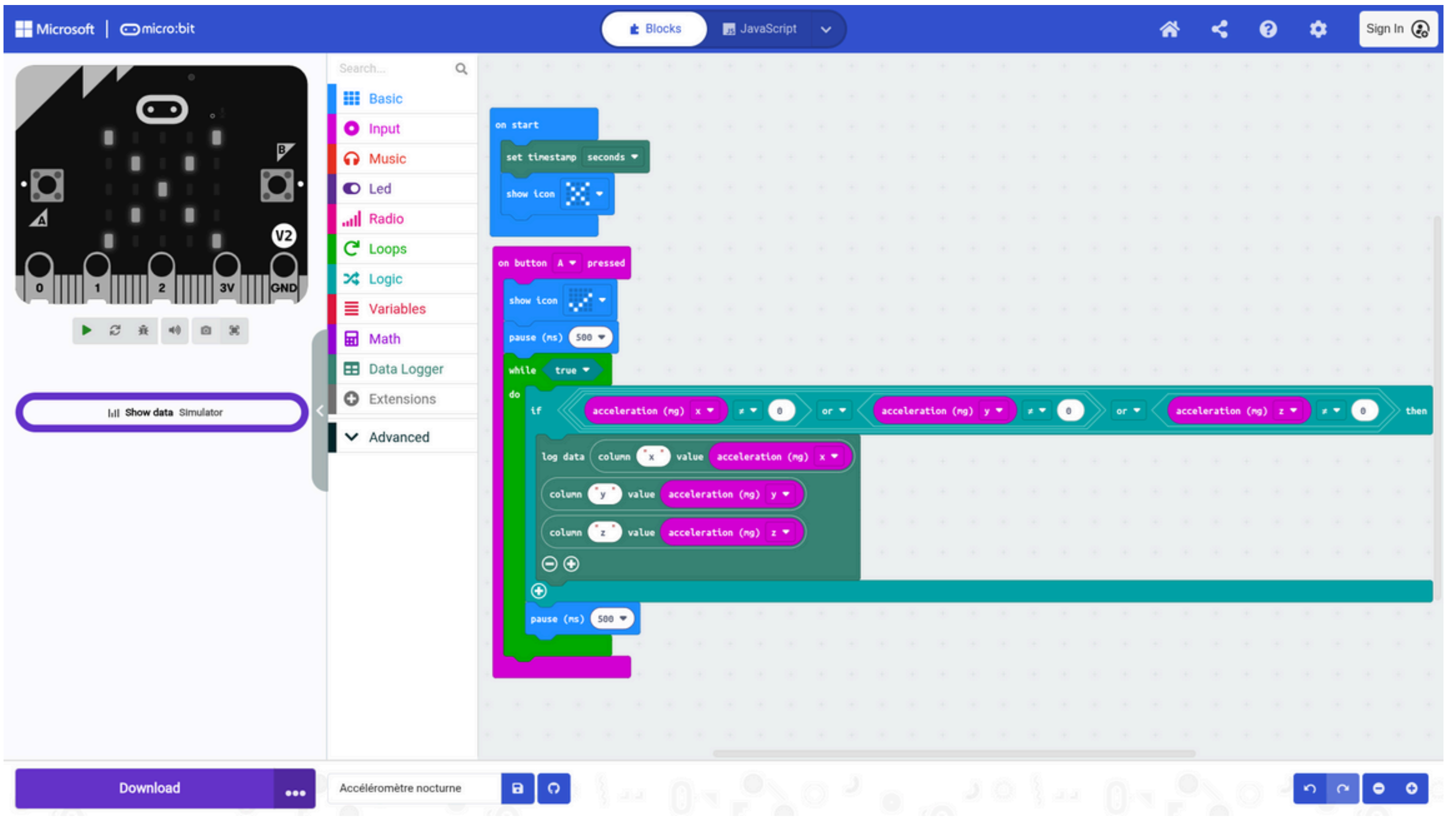
- Computer or tablet: To write and upload code.
- Programming environment: MakeCode online editor

Follow these steps to program, place, record, and retrieve environmental data using the micro:bit.

<u>Step 1 - Program the Micro:bit board</u>

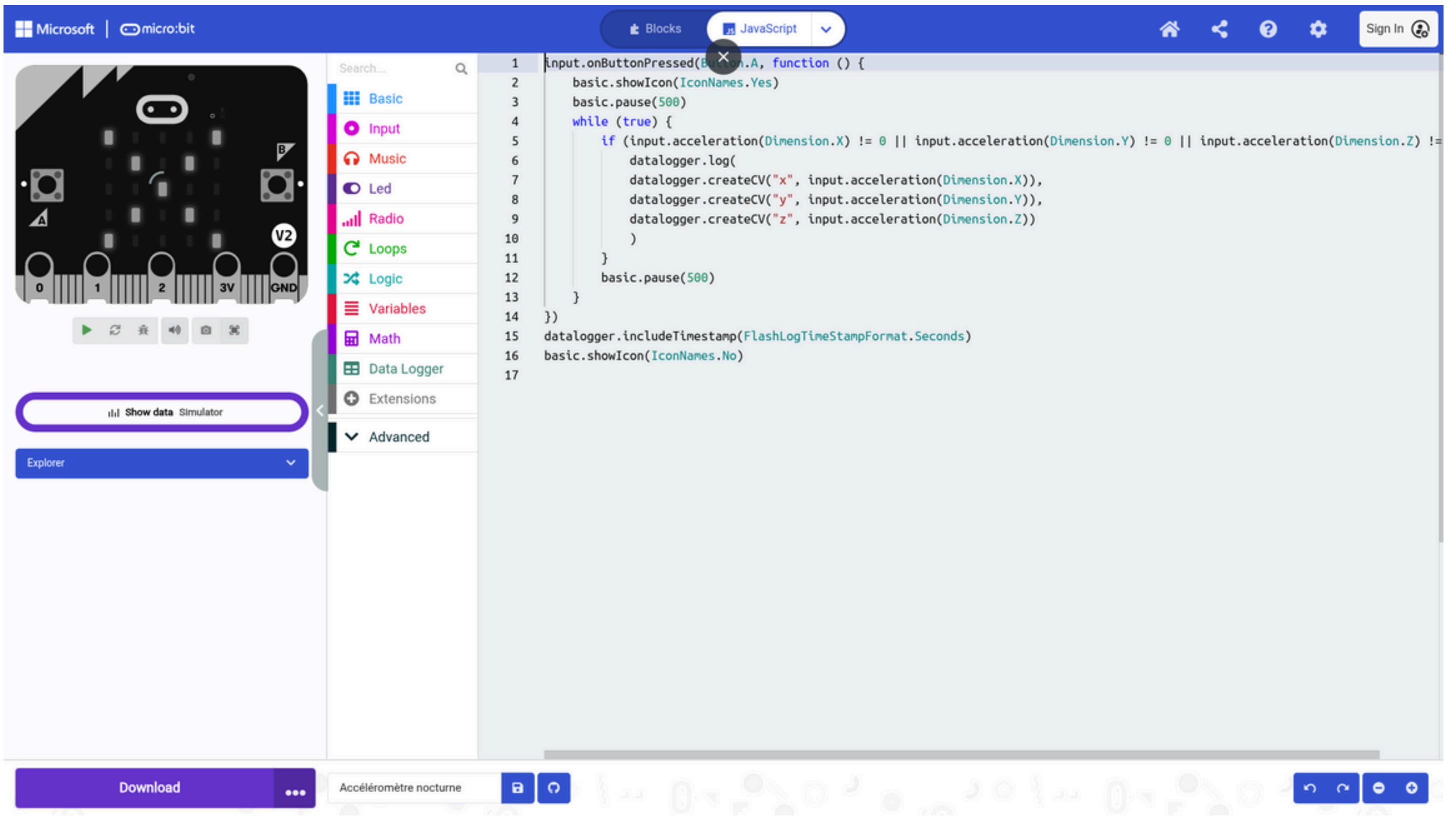
Connect the Micro:bit board: Connect the micro:bit board to the computer you created the program on using the MakeCode editor. Once connected, the micro:bit board will appear on the computer as a removable disk (e.g., "MICROBIT"). Write the program: Open the MakeCode editor to create a program that collects noise data using the built-in sensors of the Micro:bit V2 programming board. Give your project a clear name before you begin. Once in the editor, after creating your new project, you will be taken to the default "out of the box" screen and will need to install an extension. Extensions in MakeCode are groups of code blocks that are not directly included in the base MakeCode code blocks. Extensions, as the name suggests, add blocks for specific functionality. There are extensions for a wide range of very useful features, adding gamepad, keyboard, mouse, servo, and robotics capabilities, and much more. In the block display columns, click the EXTENSIONS button. In the list of available extensions, find the Datalogger extension that will be used for this activity. Click the extension you want to use and a new block group will appear on the main screen.

Then you can start organizing your blocks by following the code provided below (add an infinite loop, save the data to the datalogger...).



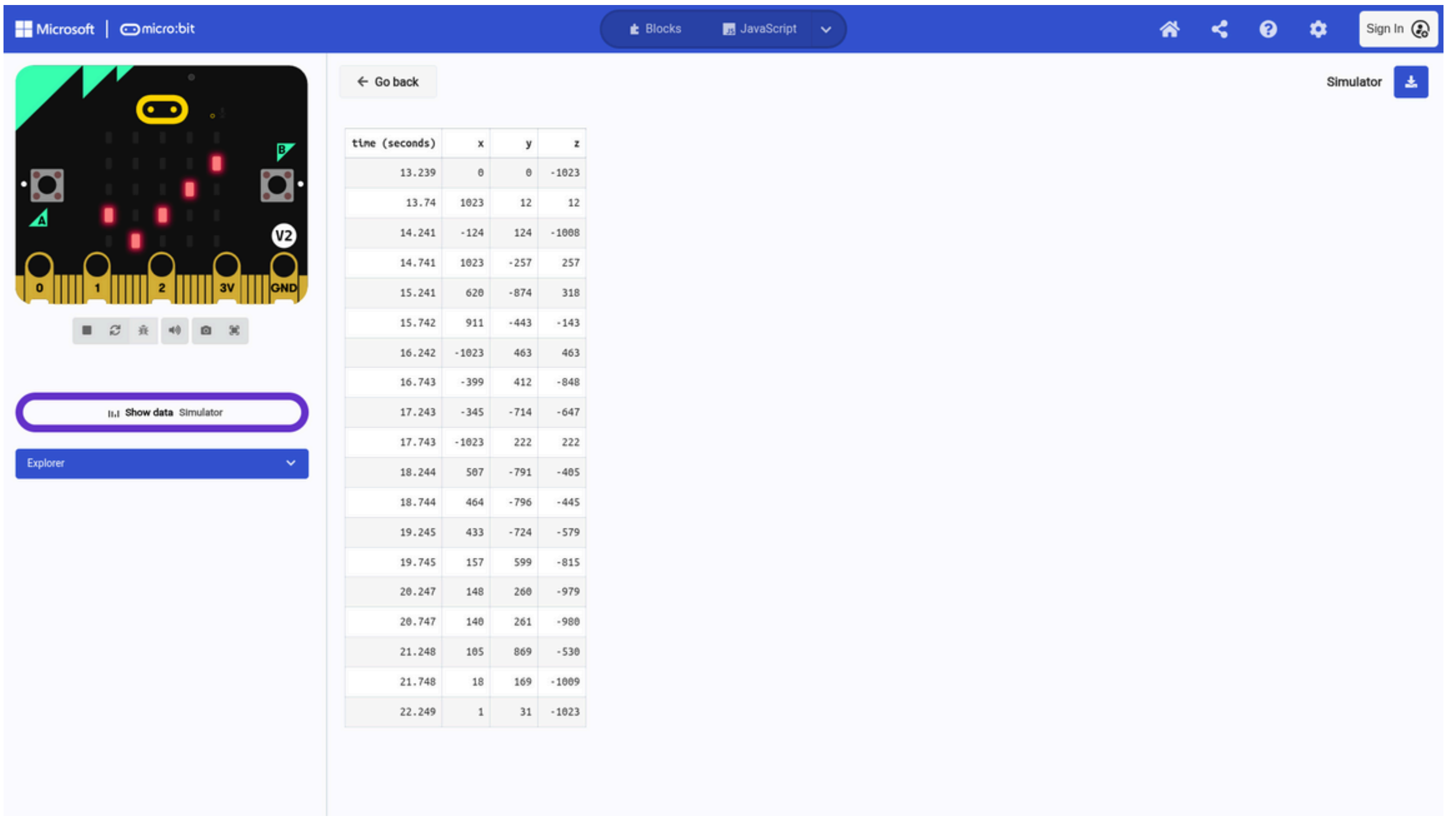
```
on start
    set timestamp seconds
    show icon

on button A pressed
    show icon
    pause (ms) 500
    while true
    do
        if acceleration (mg) x ≠ 0 or acceleration (mg) y ≠ 0 or acceleration (mg) z ≠ 0 then
            log data column "x" value acceleration (mg) x
                column "y" value acceleration (mg) y
                column "z" value acceleration (mg) z
        pause (ms) 500
```

You can also copy and paste the code into the Javascript editor.



```
input.onButtonPressed(Button.A, function () {
    basic.showIcon(IconNames.Yes)
    basic.pause(500)
    while (true) {
        if (input.acceleration(Dimension.X) != 0 || input.acceleration(Dimension.Y) != 0 || input.acceleration(Dimension.Z) !=
            datalogger.log(
            datalogger.createCV("x", input.acceleration(Dimension.X)),
            datalogger.createCV("y", input.acceleration(Dimension.Y)),
            datalogger.createCV("z", input.acceleration(Dimension.Z))
            )
        }
        basic.pause(500)
    }
})
datalogger.includeTimestamp(FlashLogTimeStampFormat.Seconds)
basic.showIcon(IconNames.No)
```

Test the program using the MakeCode simulator.



time (seconds)	x	y	z
13.239	0	0	-1023
13.74	1023	12	12
14.241	-124	124	-1008
14.741	1023	-257	257
15.241	620	-874	318
15.742	911	-443	-143
16.242	-1023	463	463
16.743	-399	412	-848
17.243	-345	-714	-647
17.743	-1023	222	222
18.244	507	-791	-405
18.744	464	-796	-445
19.245	433	-724	-579
19.745	157	599	-815
20.247	148	260	-979
20.747	140	261	-980
21.248	105	869	-530
21.748	18	169	-1009
22.249	1	31	-1023

Once your program is working correctly on the simulator, transfer it to your Micro:bit: click “Upload” in MakeCode to generate a .hex file. This file contains the compiled program that will enable the board to work.

Copy the .hex file from your download folder to the “MICROBIT” removable drive.

Once the file is copied, the board will automatically reboot and execute the code.

<u>**Step 2: Place the Micro:bit and start recording data**</u>

Once programmed, position the micro:bit to collect the data you need, using an armband. You'll find some tutorials for creating a DIY micro:bit armband at the end of this guide. Use a power bank to continuously power the micro:bit while recording (the micro:bit kit gives you access to battery boxes that are easily positioned on the board).

Before going to bed, press the "A" button on the MicroBit to start recording data via the program.

<u>**Step 3: Retrieve data and prepare the board for the next recording session**</u>

Every morning, to avoid any data loss, we recommend that you unplug the micro:bit from its power source to stop data recording and connect the micro:bit to your computer to access the file compiled overnight by the datalogger (which will be called “MY_DATA.HTM, available on the micro:bit reader).

Copiez ce fichier sur votre ordinateur et renommez-le avec la date du jour (par exemple, BOARD1_NAME_YYYY-MM-DD.HTM).

After copying and renaming the file, delete the MY_DATA.HTM file from the MicroBit board to free up space and allow for new data recording.

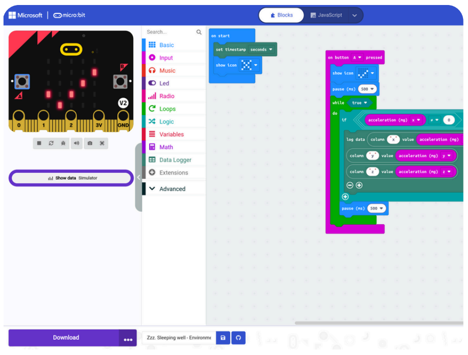
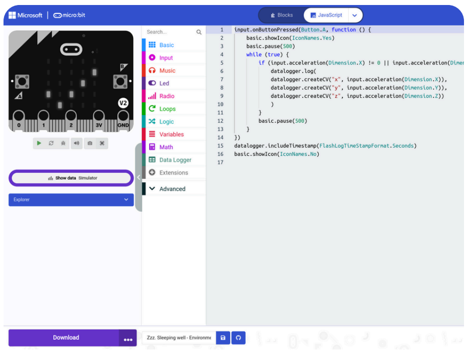
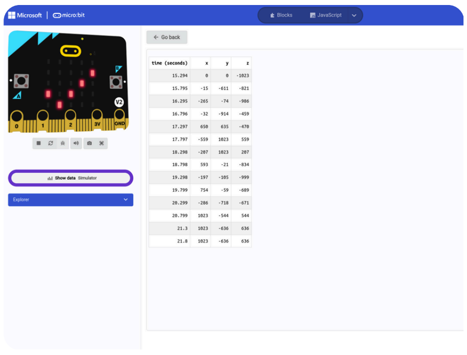
Repeat the process for the next session, that is, the next evening before going to bed.

At the end of the collection period, you will be able to retrieve all the files collected on the various micro:bit boards. Once opened, the data files will be accessible in HTML format. They will provide all the collected data and allow you to download them in .csv format.

Use and understand the code

Here is the JavaScript code used to program a micro:bit board to regularly collect motion data:

```
input.onButtonPressed(Button.A, function () {
    basic.showIcon(IconNames.Yes)
    basic.pause(500)
    while (true) {
        if (input.acceleration(Dimension.X) != 0 || input.acceleration(Dimension.Y) !=
    0 ||input.acceleration(Dimension.Z) != 0) {
            datalogger.log(
            datalogger.createCV("x", input.acceleration(Dimension.X)),
            datalogger.createCV("y", input.acceleration(Dimension.Y)),
            datalogger.createCV("z", input.acceleration(Dimension.Z))
            )
        }
        basic.pause(500)
    }
})
datalogger.includeTimestamp(FlashLogTimeStampFormat.Seconds)
basic.showIcon(IconNames.No)
```

time (seconds)	x	y	z
15.294	0	0	-1023
15.795	-15	-611	-821
16.295	-265	-74	-986
16.796	-32	-914	-459
17.297	650	635	-470
17.797	-559	1023	559
18.298	-207	1023	207
18.798	593	-21	-834
19.298	-197	-105	-999
19.799	754	-59	-689
20.299	-286	-718	-671
20.799	1023	-544	544
21.3	1023	-636	636
21.8	1023	-636	636

<u>**How does the code work?**</u>

This program measures the acceleration values captured by the accelerometer. Every 500 milliseconds (the interval can be changed to 10 seconds, 5 minutes, 2 times per hour, etc.) the program checks if the board is moving and, if so, compiles the data into a "datalogger" from which a .csv file can be downloaded.

A .csv (Comma-Separated Values) file is a text file format used to store tabular data (such as in a table or spreadsheet). Each line in the file represents a row of data, and each value within a row is separated by a delimiter (often a comma, but sometimes a semicolon or a tab). It is possible to retrieve data from a .csv file into a spreadsheet program such as Excel or LibreOffice Calc. In Excel, open the program, click File > Open, select the .csv file, and configure delimiters if necessary using the import tool. In LibreOffice Calc, follow a similar process: click File > Open, select the file, and use the import wizard to choose the delimiter (for example, comma or semicolon). In either case, the data is displayed in table format, ready for analysis.

This program is designed to record accelerometer data on a MicroBit when the "A" button is pressed and store it in a datalogger with values for the X, Y, and Z axes.

Initializing the button "A" press event: When the user presses the "A" button on the MicroBit, the function input.onButtonPressed(Button.A, function () {...}) is triggered. This prevents data from being saved as soon as the board is connected.

Displaying the "Yes" icon during execution: Before starting data recording, the program displays the "Yes" icon (basic.showIcon(IconNames.Yes)) for 500 milliseconds (0.5 seconds) to indicate that the recording process has started.

Pause for 500 milliseconds: After displaying the "Yes" icon, the program waits for 500 milliseconds using basic.pause(500).

Infinite data collection loop: The program enters an infinite while (true) loop. This means that data will be collected and recorded endlessly until the MicroBit is turned off or restarted.

Checking accelerometer data: At each iteration, the program checks whether any of the acceleration values (on the X, Y, or Z axes) are non-zero. This is done with the condition:

```
    if (input.acceleration(Dimension.X) != 0 || input.acceleration(Dimension.Y) != 0 ||
    input.acceleration(Dimension.Z) != 0)
```

If any of these values is non-zero (meaning that motion has been detected), the program records this data in the datalogger.

Data Logging: Acceleration values for the X, Y, and Z axes are logged using the datalogger.log() function: This function creates a record of the acceleration values whenever the values are non-zero, with a timestamp for each record. The timestamp is automatically added using the following line (explained later).

```
datalogger.log(
    datalogger.createCV("x", input.acceleration(Dimension.X)),
    datalogger.createCV("y", input.acceleration(Dimension.Y)),
    datalogger.createCV("z", input.acceleration(Dimension.Z))
)
```

Pause for 500 milliseconds before next reading: After saving the data, the program pauses for 500 milliseconds (basic.pause(500)) before resuming readings of the accelerometer values.

Data timestamp (included via datalogger.includeTimestamp): Apart from the button function, the datalogger.includeTimestamp(FlashLogTimeStampFormat.Seconds) command is used to include a timestamp in seconds for each record in the datalogger. This means that each record will be accompanied by the elapsed time, in seconds, since the program was started.

Displaying "No" icon before execution: Before the user presses the "A" button, the program displays a "No" icon (basic.showIcon(IconNames.No)) to indicate that the MicroBit is waiting for user action.

Data: The accelerometer returns values in milli-g for each axis (X, Y, Z). This means that if the accelerometer detects an acceleration of 1000 milli-g, this corresponds to an acceleration of 1G (or 9.81 m/s^2) on that axis.

Add an armband

There are several resources available to help you attach the Micro:bit board to an armband, allowing students to wear it overnight. Here are some helpful tutorials and accessories to help you with this step:

Smart Coding Watch Kit - micro:bit

Duct Tape Watch

BBC micro:bit wrist holder | mattoppenheim

Yahboom Wrist:bit wearable watch kit based on BBC Micro:bit V2/V1.5 board

CHARGE for micro:bit