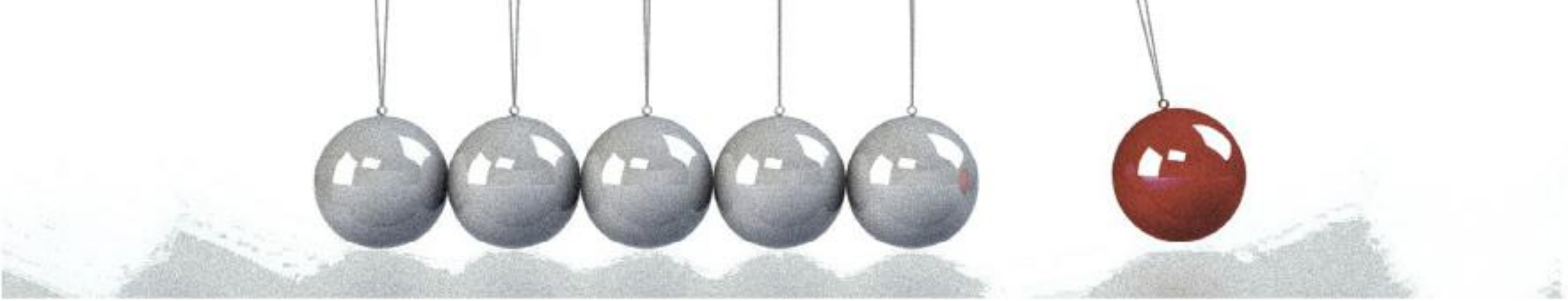


UiO : Department of Physics
University of Oslo

FYS3240- 4240
Data acquisition & control

Arduino and Arduino Nano 33 BLE sense

Spring 2021– Lecture #5

Bekkeng 12.01.2021

What is Arduino:

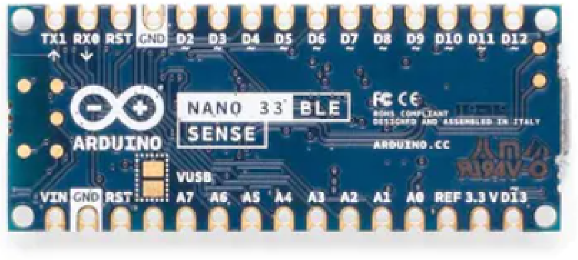
- Arduino is an open-source electronics platform based on easy-to-use hardware and software

Why Arduino in the course?

- Easy to get stated!
  - Many examples available
  - **Ready made interfaces to sensors**
  - **Data communication with a computer is easy**
  - Call C/C++ functions
  - Easy-to-use for beginners, but flexible enough for advanced users.

- Can be used for professional applications
  - But the Arduino environment hide much details about what’s under the hood.
  - The course focus is <u>not</u> microcontroller programming alone, but the collection of sensor data, processing of sensor data and how they are used in a system with feedback (control).

Nano 33 BLE Sense

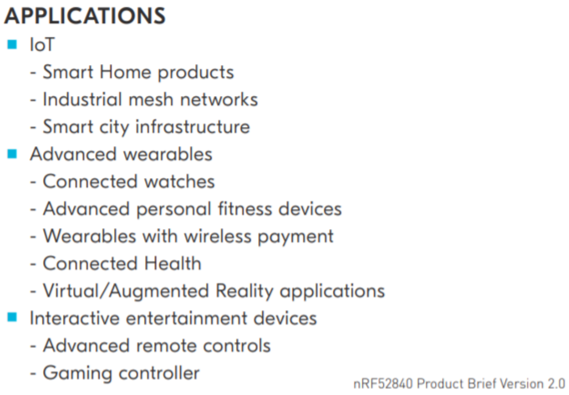


Estimate shipping date, mid August 2019.

This compact and reliable Nano board is built around the NINA B306 module for BLE and Bluetooth 5 communication; the module is based on Nordic nRF 52840 processor that contains a powerful Cortex M4F and the board has a rich set of sensors that allow the creation of innovative and highly interactive designs.

Its architecture, fully compatible with Arduino IDE Online and Offline, has a 9 axis Inertial Measurement Unit (IMU), temperature, pressure, humidity, light, color and even gestures sensors and a microphone that are managed through our specialised libraries. Its reduced power consumption, compared to other same size boards, together with the Nano form factor opens up a wide range of applications.

This allows the design of wearable devices and gesture based projects that need to communicate to other devices at a close range. Arduino Nano 33 BLE Sense is ideal for interactive automation projects thanks to the multiprotocol BT 5.0 radio.



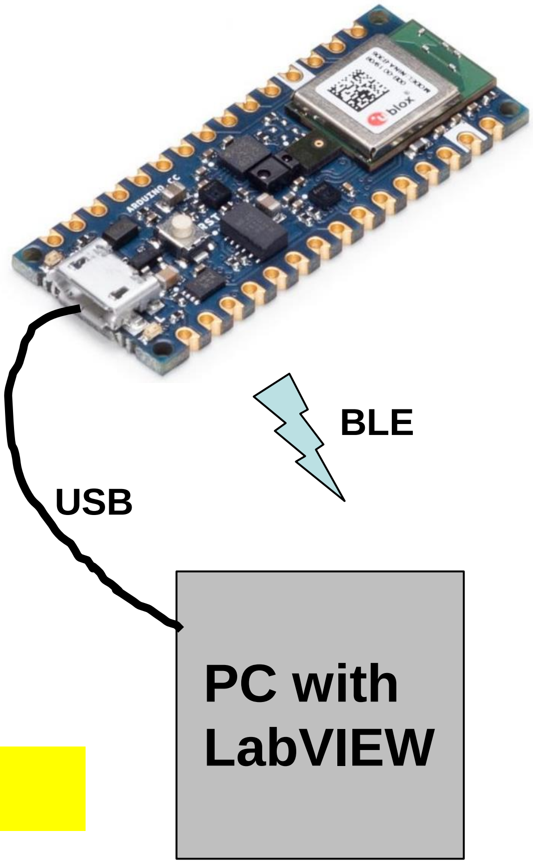
APPLICATIONS

- IoT
  - Smart Home products
  - Industrial mesh networks
  - Smart city infrastructure
- Advanced wearables
  - Connected watches
  - Advanced personal fitness devices
  - Wearables with wireless payment
  - Connected Health
  - Virtual/Augmented Reality applications
- Interactive entertainment devices
  - Advanced remote controls
  - Gaming controller

nRF52840 Product Brief Version 2.0

We will use Arduino Nano 33 BLE sense

- **ARM Cortex M4**
- **Inertial measurement unit (IMU)**
  - **3-axis accelerometer**
  - **3-axis rate gyroscope**
- **3-axis magnetometer**
- **Bluetooth low energy (BLE)**
- + many other sensors



BLE

USB

PC with LabVIEW

Small size, low cost, powerful and many sensors!

https://www.arduino.cc/en/Guide/NANO33BLESense

Nano 33 BLE sense

If you get an error or unexpected behaviour (for instance not able to upload sketch), try to press the reset switch

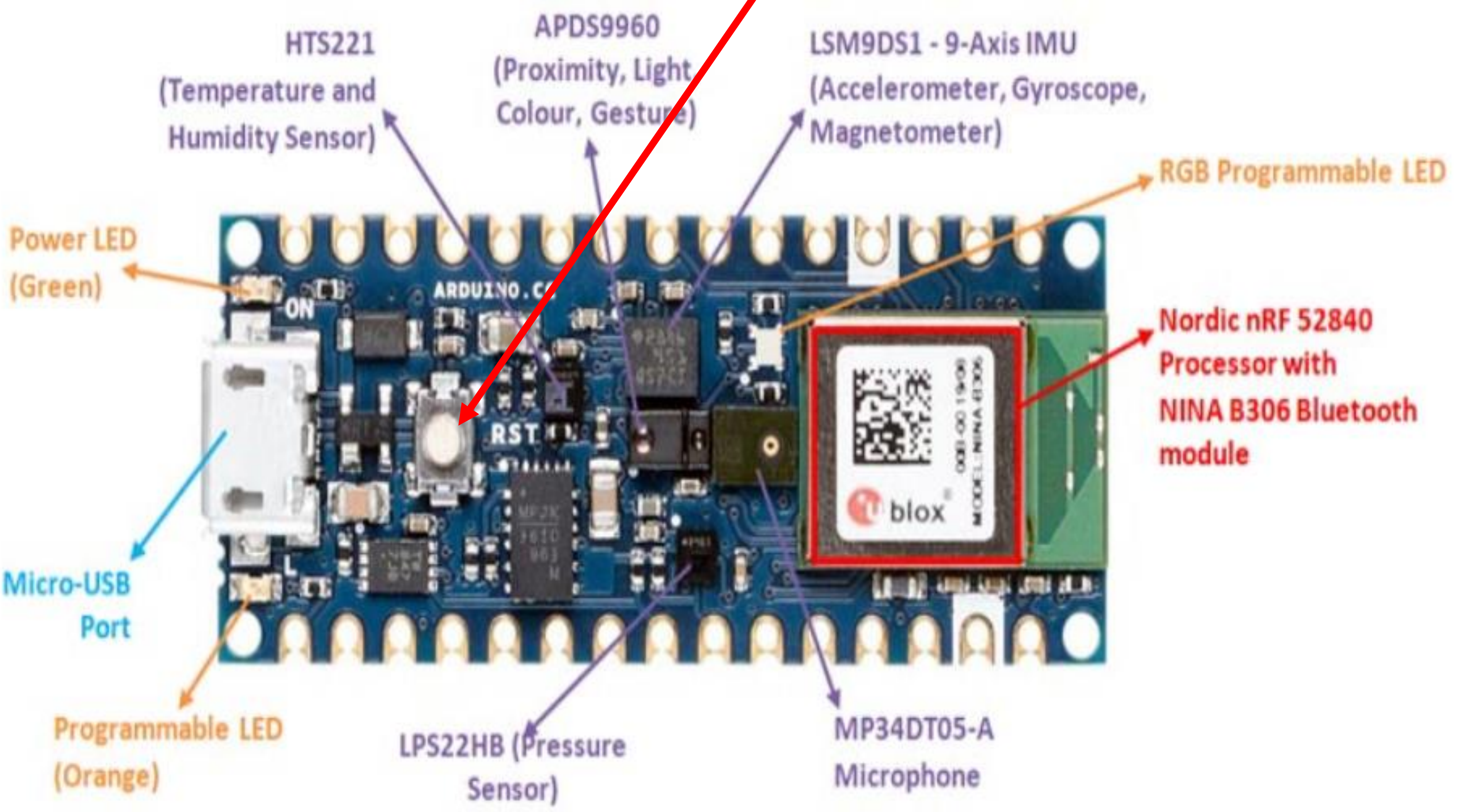


Figure from [Arduino Nano 33 BLE Sense Review - What's New and How to Get Started? (circuitdigest.com)]

ARM® Cortex®-M

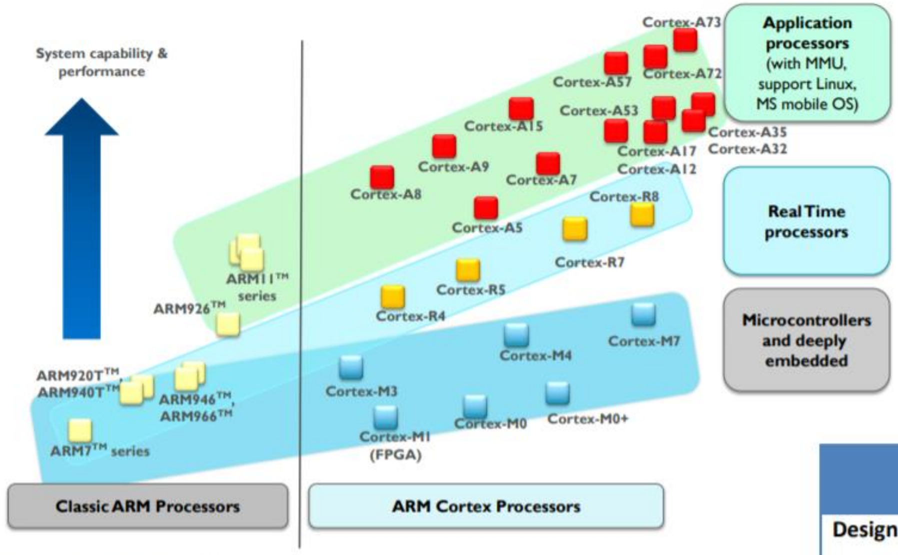
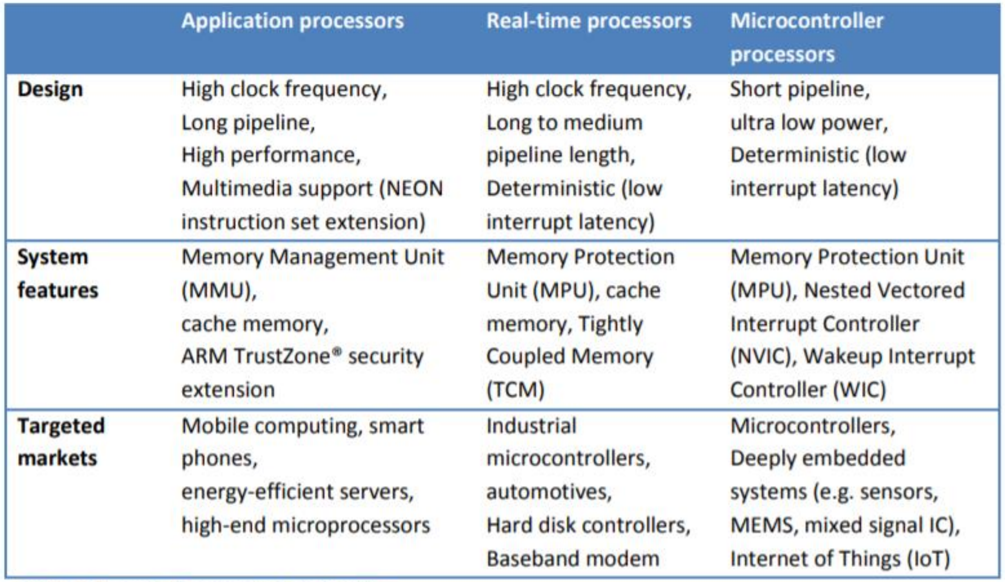


Figure 1: ARM processor family

The criteria of selecting a microcontroller device are mostly heavily dependent on the cost and peripherals available.



	Application processors	Real-time processors	Microcontroller processors
Design	High clock frequency, Long pipeline, High performance, Multimedia support (NEON instruction set extension)	High clock frequency, Long to medium pipeline length, Deterministic (low interrupt latency)	Short pipeline, ultra low power, Deterministic (low interrupt latency)
System features	Memory Management Unit (MMU), cache memory, ARM TrustZone® security extension	Memory Protection Unit (MPU), cache memory, Tightly Coupled Memory (TCM)	Memory Protection Unit (MPU), Nested Vectored Interrupt Controller (NVIC), Wakeup Interrupt Controller (WIC)
Targeted markets	Mobile computing, smart phones, energy-efficient servers, high-end microprocessors	Industrial microcontrollers, automotives, Hard disk controllers, Baseband modem	Microcontrollers, Deeply embedded systems (e.g. sensors, MEMS, mixed signal IC), Internet of Things (IoT)

Table 1: Summary of processor characteristics

Suggested reading:
White paper "ARM® Cortex®-M for Beginners", particularly section 1 to 2.4.

ARM® Cortex®-M

- Almost everything can be programmed in high-level language like C
  - easy to use
- The consistency of the architecture make it easier to start using a new Cortex-M processor once you have experience with one of them.

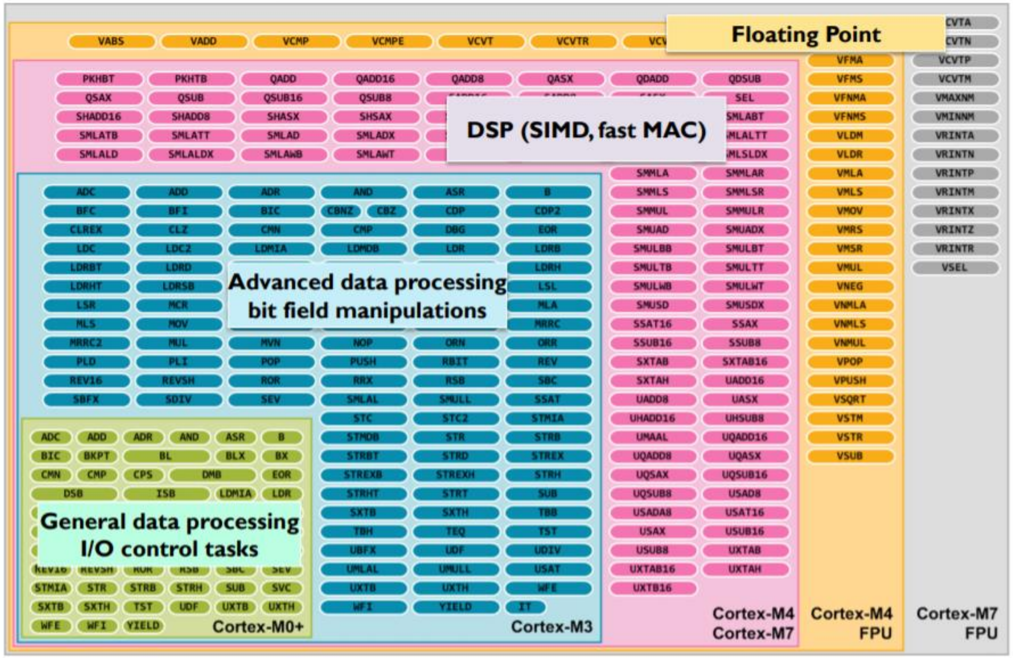
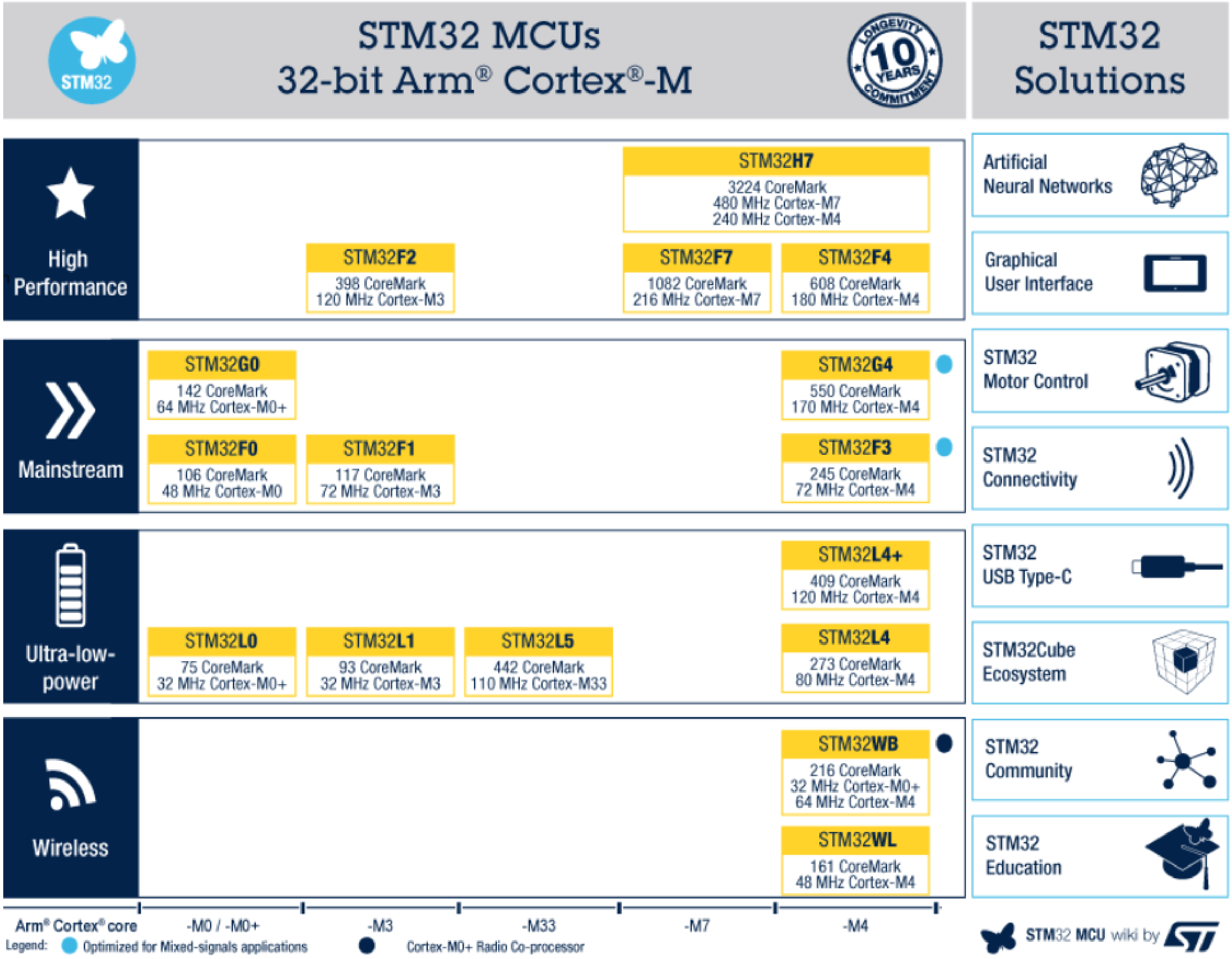


Figure 3: Instruction Set support in the Cortex-M processors

Optional:
See the White paper "ARM® Cortex®-M for Beginners"



STM32 MCUs
32-bit Arm® Cortex®-M

LONGEVITY 10 YEARS COMMITMENT

High Performance

- STM32H7: 3224 CoreMark, 480 MHz Cortex-M7, 240 MHz Cortex-M4
- STM32F2: 398 CoreMark, 120 MHz Cortex-M3
- STM32F7: 1082 CoreMark, 216 MHz Cortex-M7
- STM32F4: 608 CoreMark, 180 MHz Cortex-M4

Mainstream

- STM32G0: 142 CoreMark, 64 MHz Cortex-M0+
- STM32G4 ●: 550 CoreMark, 170 MHz Cortex-M4
- STM32F0: 106 CoreMark, 48 MHz Cortex-M0
- STM32F1: 117 CoreMark, 72 MHz Cortex-M3
- STM32F3 ●: 245 CoreMark, 72 MHz Cortex-M4

Ultra-low-power

- STM32L4+: 409 CoreMark, 120 MHz Cortex-M4
- STM32L0: 75 CoreMark, 32 MHz Cortex-M0+
- STM32L1: 93 CoreMark, 32 MHz Cortex-M3
- STM32L5: 442 CoreMark, 110 MHz Cortex-M33
- STM32L4: 273 CoreMark, 80 MHz Cortex-M4

Wireless

- STM32WB ●: 216 CoreMark, 32 MHz Cortex-M0+, 64 MHz Cortex-M4
- STM32WL: 161 CoreMark, 48 MHz Cortex-M4

Arm® Cortex® core: -M0 / -M0+, -M3, -M33, -M7, -M4

Legend: ● Optimized for Mixed-signals applications ● Cortex-M0+ Radio Co-processor

STM32 Solutions

- Artificial Neural Networks
- Graphical User Interface
- STM32 Motor Control
- STM32 Connectivity
- STM32 USB Type-C
- STM32Cube Ecosystem
- STM32 Community
- STM32 Education

STM32 MCU wiki by ST

https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html

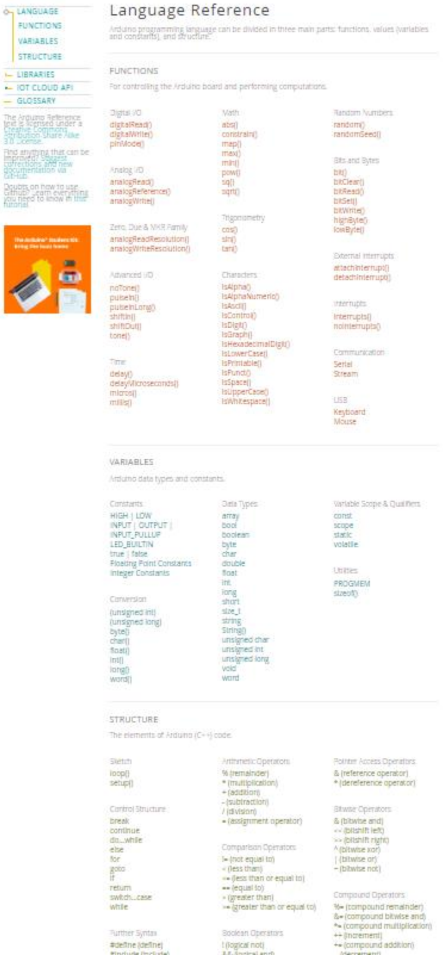
Arduino code example

- Take a value in between 0 and 1023 (e.g. 10 bit input from an ADC) and map it with in the 0 to 100 range:

```
int value2 = map(value, 0, 1023, 0, 100);
```

Arduino language references

- https://www.arduino.cc/reference/en/



LANGUAGE
FUNCTIONS
VARIABLES
STRUCTURE
LIBRARIES
IOT CLOUD API
GLOSSARY

The Arduino Reference text is licensed under a Creative Commons Attribution-Share Alike 3.0 License.

Find anything that can be improved? Suggest corrections and new documentation via GitHub.

Doubts on how to use Github? Learn everything you need to know in this tutorial.

Language Reference

Arduino programming language can be divided in three main parts: functions, values (variables and constants), and structure.

FUNCTIONS

For controlling the Arduino board and performing computations.

Digital I/O
digitalRead()
digitalWrite()
pinMode()

Analog I/O
analogRead()
analogReference()
analogWrite()

Zero, Due & MKR Family
analogReadResolution()
analogWriteResolution()

Advanced I/O
noTone()
pulseIn()
pulseInLong()
shiftIn()
shiftOut()
tone()

Time
delay()
delayMicroseconds()
micros()
millis()

Math
abs()
constrain()
map()
max()
min()
pow()
sq()
sqrt()

Trigonometry
cos()
sin()
tan()

Characters
isAlpha()
isAlphaNumeric()
isAscii()
isControl()
isDigit()
isGraph()
isHexadecimalDigit()
isLowerCase()
isPrintable()
isPunct()
isSpace()
isUpperCase()
isWhitespace()

Random Numbers
random()
randomSeed()

Bits and Bytes
bit()
bitClear()
bitRead()
bitSet()
bitWrite()
highByte()
lowByte()

External Interrupts
attachInterrupt()
detachInterrupt()

Interrupts
interrupts()
noInterrupts()

Communication
Serial
Stream

USB
Keyboard
Mouse

VARIABLES

Arduino data types and constants.

Constants
HIGH | LOW
INPUT | OUTPUT | INPUT_PULLUP
LED_BUILTIN
true | false
Floating Point Constants
Integer Constants

Conversion
(unsigned int)
(unsigned long)
byte()
char()
float()
int()
long()
word()

Data Types
array
bool
boolean
byte
char
double
float
int
long
short
size_t
string
String()
unsigned char
unsigned int
unsigned long
void
word

Variable Scope & Qualifiers
const
scope
static
volatile

Utilities
PROGMEM
sizeof()

STRUCTURE

The elements of Arduino (C++) code.

Sketch
loop()
setup()

Control Structure
break
continue
do...while
else
for
goto
if
return
switch...case
while

Further Syntax
#define (define)
#include (include)

Arithmetic Operators
% (remainder)
* (multiplication)
+ (addition)
- (subtraction)
/ (division)
= (assignment operator)

Comparison Operators
!= (not equal to)
< (less than)
<= (less than or equal to)
== (equal to)
> (greater than)
>= (greater than or equal to)

Boolean Operators
! (logical not)
&& (logical and)

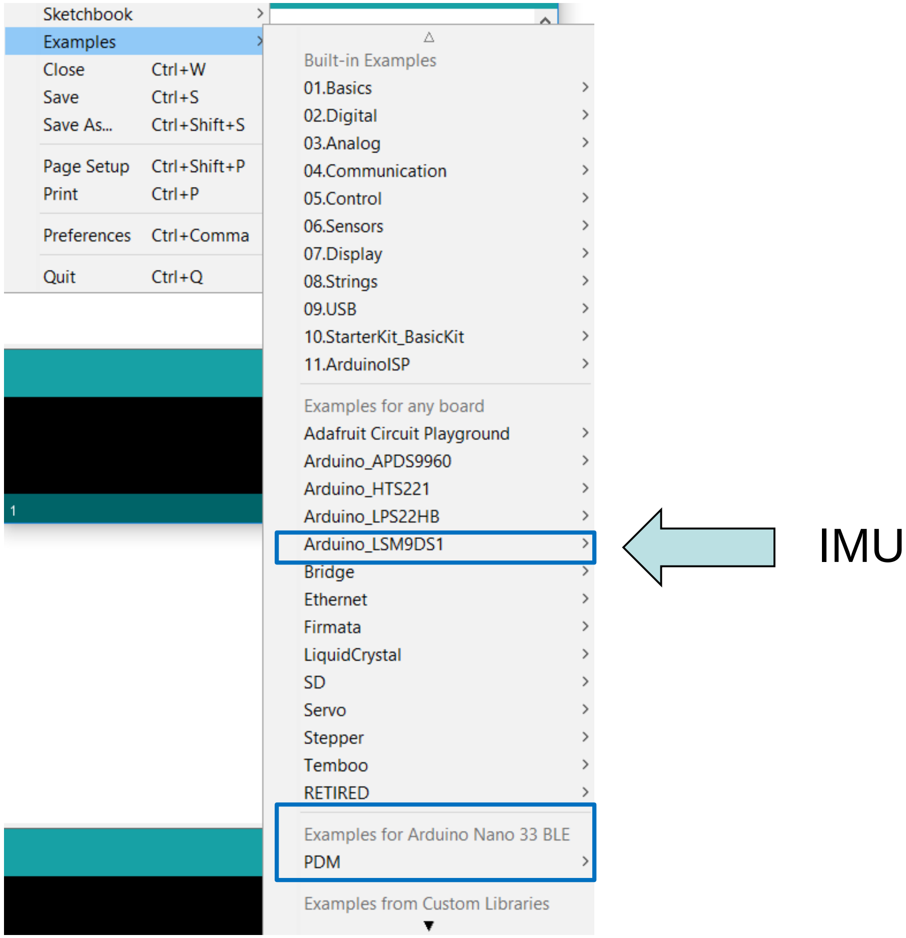
Pointer Access Operators
& (reference operator)
* (dereference operator)

Bitwise Operators
& (bitwise and)
<< (bitshift left)
>> (bitshift right)
^ (bitwise xor)
| (bitwise or)
~ (bitwise not)

Compound Operators
%= (compound remainder)
&= (compound bitwise and)
*= (compound multiplication)
++ (increment)
+= (compound addition)
-- (decrement)

Examples for Nano 33 BLE sense in IDE

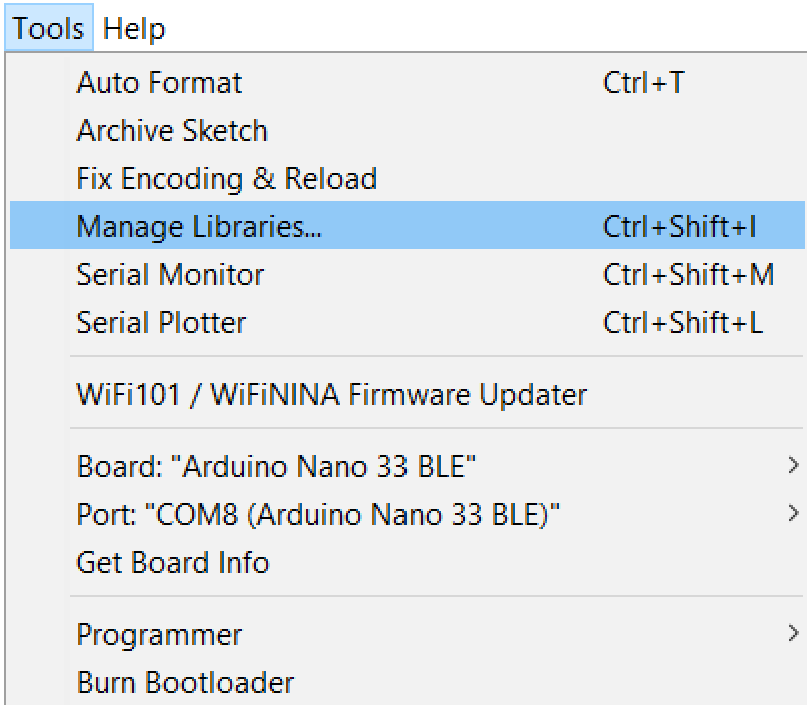
Under Files:

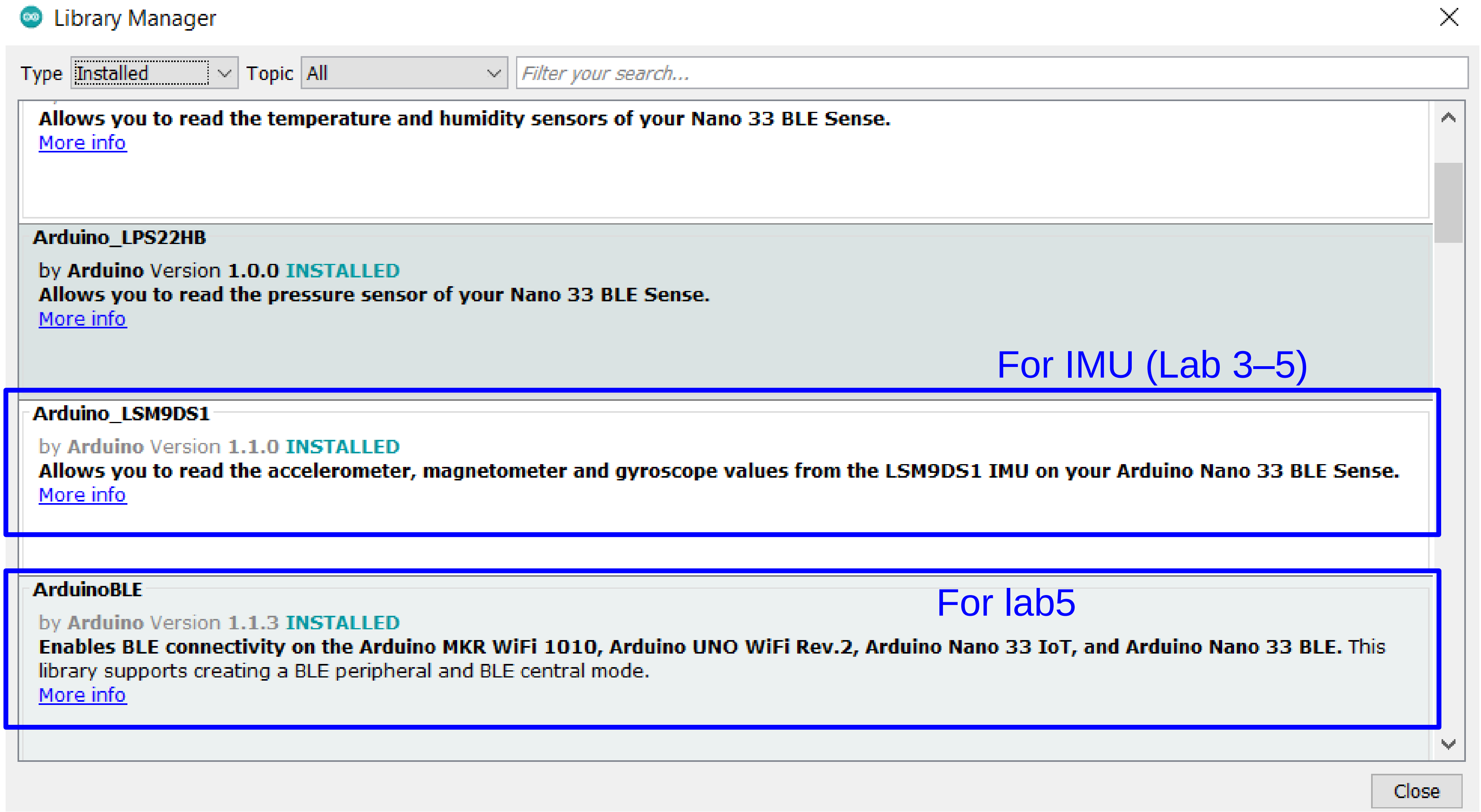


Arduino libraries

- https://www.arduino.cc/en/Guide/Libraries#toc3

In Arduino IDE:


Tools
Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Nano 33 BLE"
Port: "COM8 (Arduino Nano 33 BLE)"
Get Board Info
Programmer
Burn Bootloader



Allows you to read the temperature and humidity sensors of your Nano 33 BLE Sense.
More info

Arduino_LPS22HB
by **Arduino** Version **1.0.0** **INSTALLED**
Allows you to read the pressure sensor of your Nano 33 BLE Sense.
More info

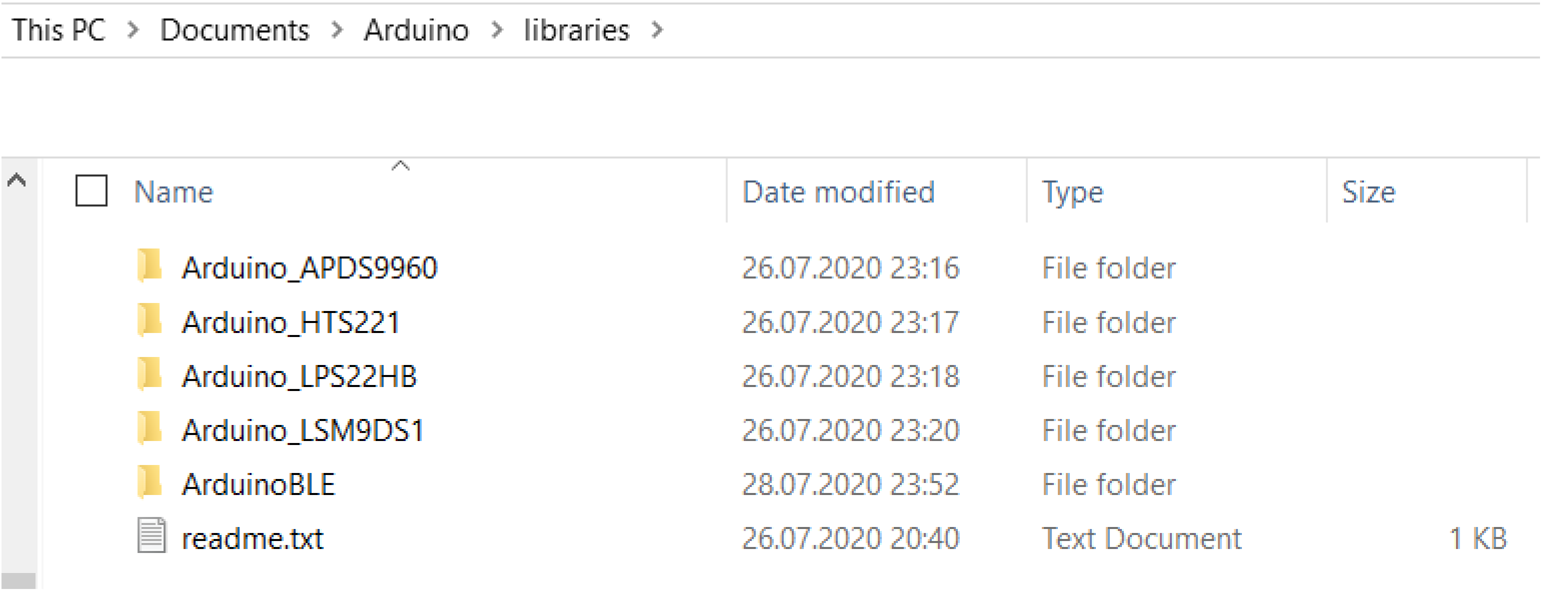
For IMU (Lab 3–5)

Arduino_LSM9DS1
by **Arduino** Version **1.1.0** **INSTALLED**
Allows you to read the accelerometer, magnetometer and gyroscope values from the LSM9DS1 IMU on your Arduino Nano 33 BLE Sense.
More info

ArduinoBLE
by **Arduino** Version **1.1.3** **INSTALLED**
Enables BLE connectivity on the Arduino MKR WiFi 1010, Arduino UNO WiFi Rev.2, Arduino Nano 33 IoT, and Arduino Nano 33 BLE. This library supports creating a BLE peripheral and BLE central mode.
More info

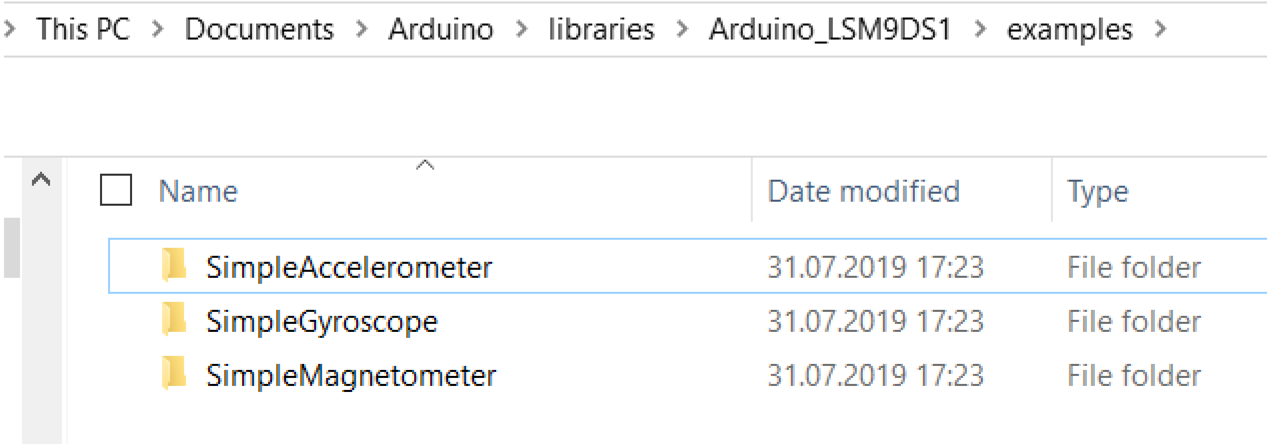
For lab5

Arduino libraries



This PC > Documents > Arduino > libraries >

□ Name	Date modified	Type	Size
Arduino_APDS9960	26.07.2020 23:16	File folder	
Arduino_HTS221	26.07.2020 23:17	File folder	
Arduino_LPS22HB	26.07.2020 23:18	File folder	
Arduino_LSM9DS1	26.07.2020 23:20	File folder	
ArduinoBLE	28.07.2020 23:52	File folder	
readme.txt	26.07.2020 20:40	Text Document	1 KB

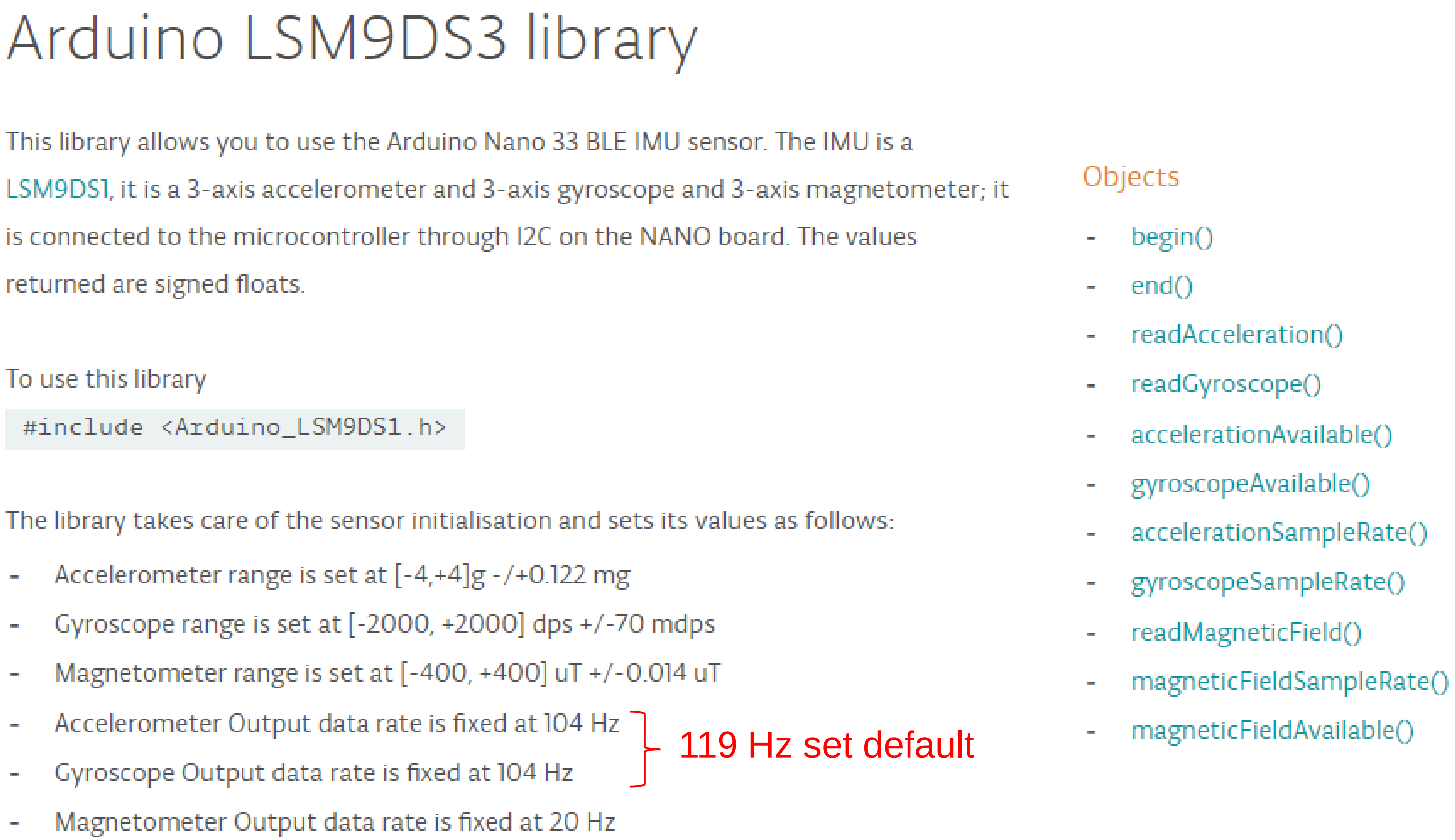


> This PC > Documents > Arduino > libraries > Arduino_LSM9DS1 > examples >

□ Name	Date modified	Type
SimpleAccelerometer	31.07.2019 17:23	File folder
SimpleGyroscope	31.07.2019 17:23	File folder
SimpleMagnetometer	31.07.2019 17:23	File folder

IMU an magnetometer library for the Nano 33 BLE sense board

- https://www.arduino.cc/en/Reference/ArduinoLSM9DS1



Arduino LSM9DS3 library

This library allows you to use the Arduino Nano 33 BLE IMU sensor. The IMU is a LSM9DS1, it is a 3-axis accelerometer and 3-axis gyroscope and 3-axis magnetometer; it is connected to the microcontroller through I2C on the NANO board. The values returned are signed floats.

To use this library

```
#include <Arduino_LSM9DS1.h>
```

The library takes care of the sensor initialisation and sets its values as follows:

- Accelerometer range is set at [-4,+4]g -/+0.122 mg
- Gyroscope range is set at [-2000, +2000] dps +/-70 mdps
- Magnetometer range is set at [-400, +400] uT +/-0.014 uT
- Accelerometer Output data rate is fixed at 104 Hz
- Gyroscope Output data rate is fixed at 104 Hz
- Magnetometer Output data rate is fixed at 20 Hz

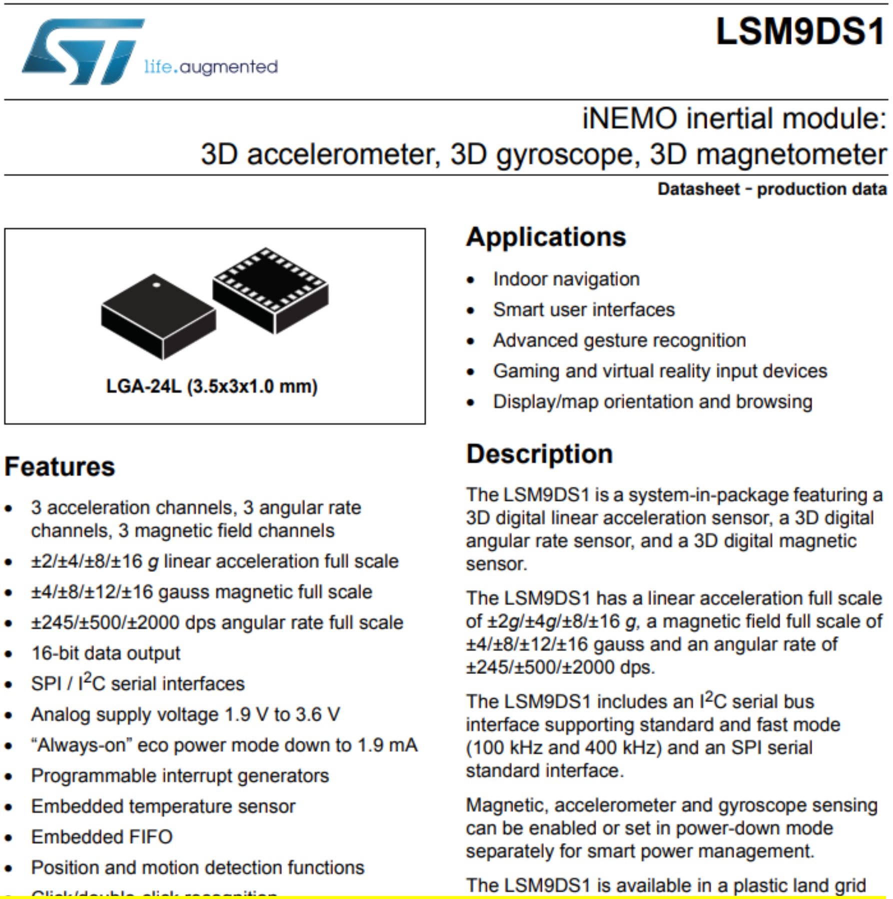
119 Hz set default

Objects

- begin()
- end()
- readAcceleration()
- readGyroscope()
- accelerationAvailable()
- gyroscopeAvailable()
- accelerationSampleRate()
- gyroscopeSampleRate()
- readMagneticField()
- magneticFieldSampleRate()
- magneticFieldAvailable()

UiO : **Department of Physics**
University of Oslo

LSM9DS1 data sheet



LSM9DS1

life.augmented

iNEMO inertial module:
3D accelerometer, 3D gyroscope, 3D magnetometer

Datasheet - production data

LGA-24L (3.5x3x1.0 mm)

Features

- 3 acceleration channels, 3 angular rate channels, 3 magnetic field channels
- ±2/±4/±8/±16 *g* linear acceleration full scale
- ±4/±8/±12/±16 gauss magnetic full scale
- ±245/±500/±2000 dps angular rate full scale
- 16-bit data output
- SPI / I^{2}C serial interfaces
- Analog supply voltage 1.9 V to 3.6 V
- “Always-on” eco power mode down to 1.9 mA
- Programmable interrupt generators
- Embedded temperature sensor
- Embedded FIFO
- Position and motion detection functions
- Click/double-click recognition

Applications

- Indoor navigation
- Smart user interfaces
- Advanced gesture recognition
- Gaming and virtual reality input devices
- Display/map orientation and browsing

Description

The LSM9DS1 is a system-in-package featuring a 3D digital linear acceleration sensor, a 3D digital angular rate sensor, and a 3D digital magnetic sensor.

The LSM9DS1 has a linear acceleration full scale of ±2*g*/±4*g*/±8/±16 *g*, a magnetic field full scale of ±4/±8/±12/±16 gauss and an angular rate of ±245/±500/±2000 dps.

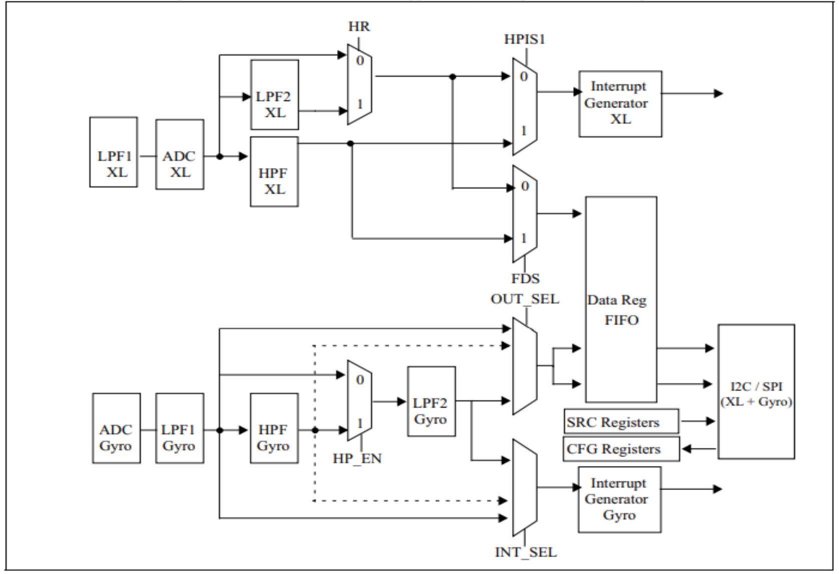
The LSM9DS1 includes an I^{2}C serial bus interface supporting standard and fast mode (100 kHz and 400 kHz) and an SPI serial standard interface.

Magnetic, accelerometer and gyroscope sensing can be enabled or set in power-down mode separately for smart power management.

The LSM9DS1 is available in a plastic land grid

Have a look! Without the Arduino library you had to know “all of this” to implement communication with the IMU

Figure 8. Accelerometer and gyroscope digital block diagram



LSM9DS1 Digital interfaces

5.2 Accelerometer and gyroscope SPI bus interface

The LSM9DS1 accelerometer and gyroscope SPI is a bus slave. The SPI allows to write and read the registers of the device.

The Serial Interface connects to applications using 4 wires: **CS_A/G**, **SPC**, **SDI** and **SDO_A/G**.

Figure 16. Accelerometer and gyroscope read and write protocol

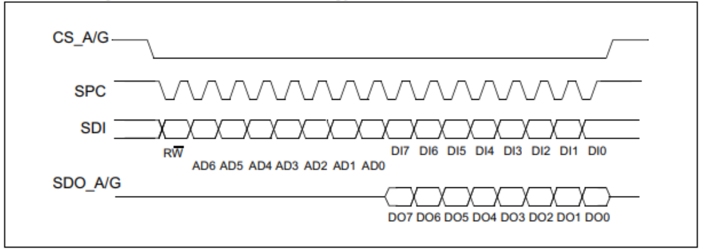

CS_A/G
SPC
SDI
RW
AD6 AD5 AD4 AD3 AD2 AD1 AD0
DI7 DI6 DI5 DI4 DI3 DI2 DI1 DI0
SDO_A/G
DO7 DO6 DO5 DO4 DO3 DO2 DO1 DO0

Table 21. Accelerometer and gyroscope register address map (continued)

Name	Type	Register address		Default	Note
		Hex	Binary		
CTRL_REG6_XL	r/w	20	00100000	00000000	
CTRL_REG7_XL	r/w	21	00100001	00000000	
CTRL_REG8	r/w	22	00100010	00000100	
CTRL_REG9	r/w	23	00100011	00000000	
CTRL_REG10	r/w	24	00100100	00000000	

Float vs. integer vs. string.

We need to understand the basics:

- Float (double) vs. int.
- Floating point (decimal) numbers vs. fixed point number

Assumed known, or google! See also Arduino cookbook

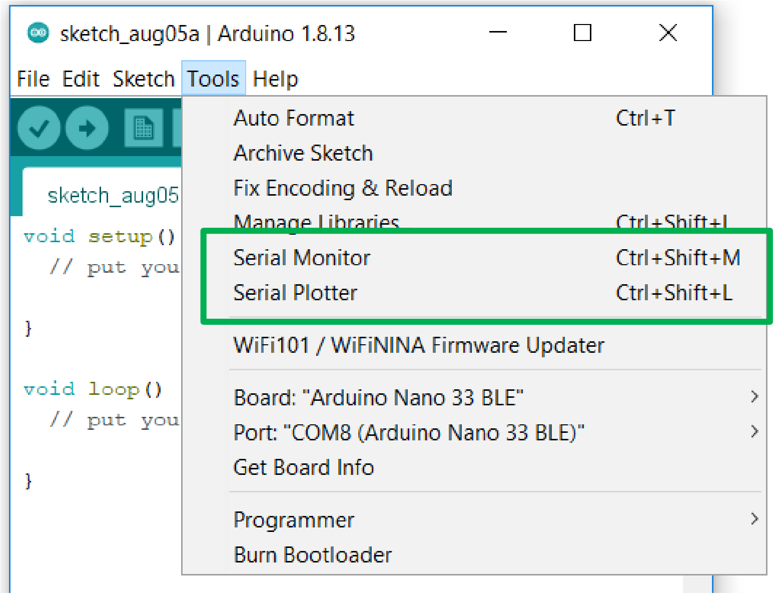
- Digital sensors deliver data as integers (2^n) !

- If we measure e.g. voltage or rotation rate we need to get a decimal number before using the data
  - 3.14 V
  - 20.32 deg/sec

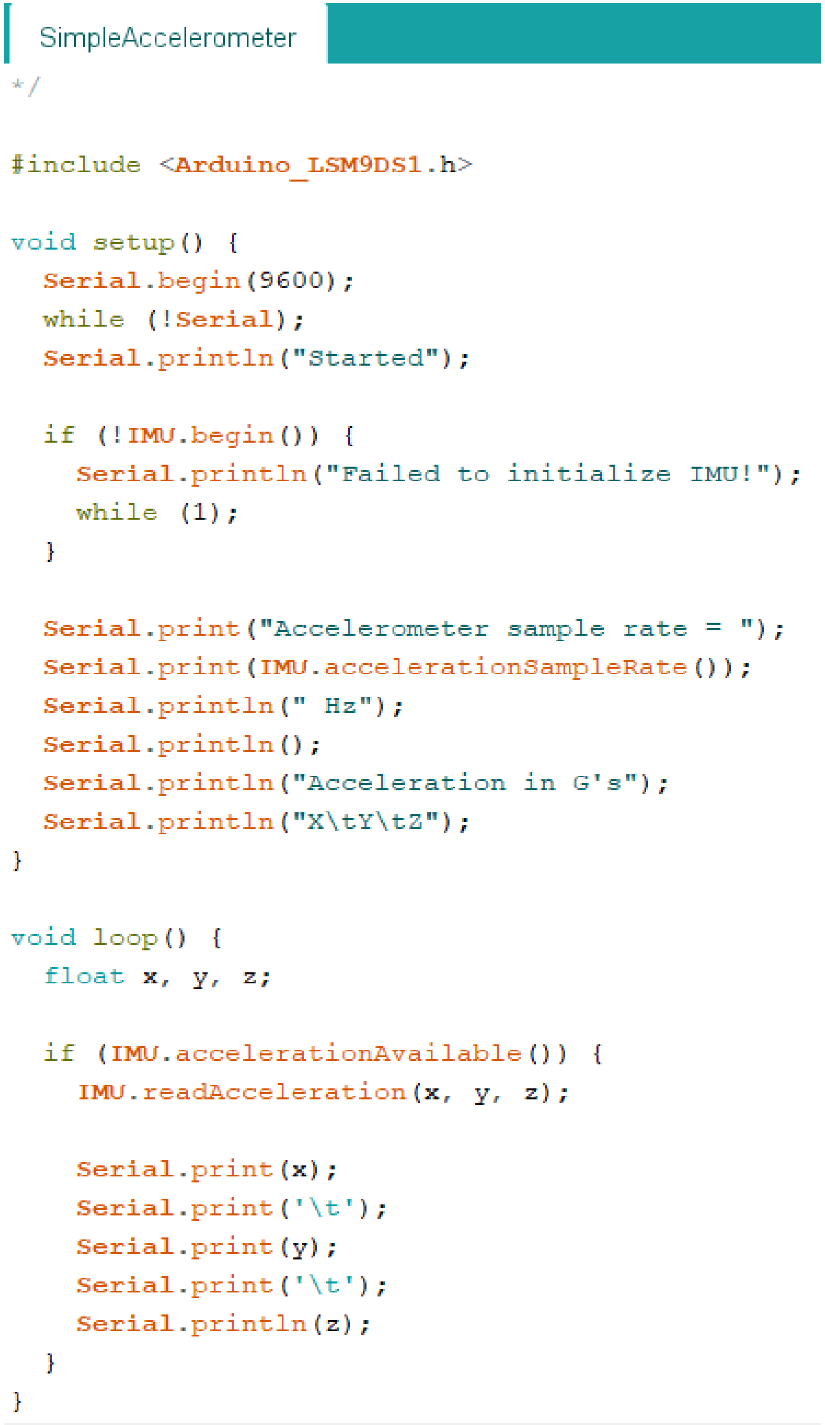
In this course we will send sensor data as float values converted to strings. This is not the most efficient way, but it is easy to write/read and to debug.

Accelerometer example

This example reads the acceleration values from the LSM9DS1 sensor and continuously prints them to the Serial Monitor or Serial Plotter in the IDE

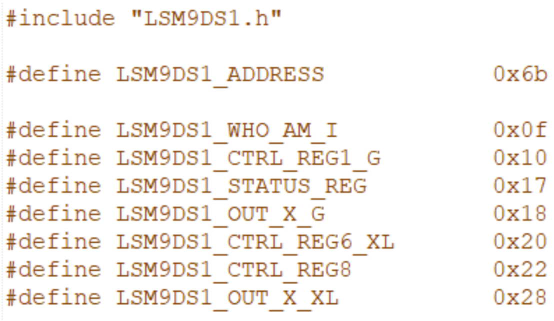

sketch_aug05a | Arduino 1.8.13
File Edit Sketch Tools Help
sketch_aug05
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Nano 33 BLE"
Port: "COM8 (Arduino Nano 33 BLE)"
Get Board Info
Programmer
Burn Bootloader

```
void setup()
  // put you

}

void loop()
  // put you

}
```

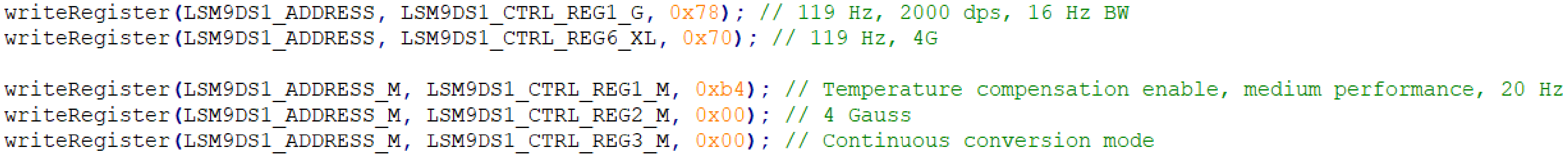


```
*/

#include <Arduino_LSM9DS1.h>

void setup() {
  Serial.begin(9600);
  while (!Serial);
  Serial.println("Started");

  if (!IMU.begin()) {
    Serial.println("Failed to initialize IMU!");
    while (1);
  }

  Serial.print("Accelerometer sample rate = ");
  Serial.print(IMU.accelerationSampleRate());
  Serial.println(" Hz");
  Serial.println();
  Serial.println("Acceleration in G's");
  Serial.println("X\tY\tZ");
}

void loop() {
  float x, y, z;

  if (IMU.accelerationAvailable()) {
    IMU.readAcceleration(x, y, z);

    Serial.print(x);
    Serial.print('\t');
    Serial.print(y);
    Serial.print('\t');
    Serial.println(z);
  }
}
```

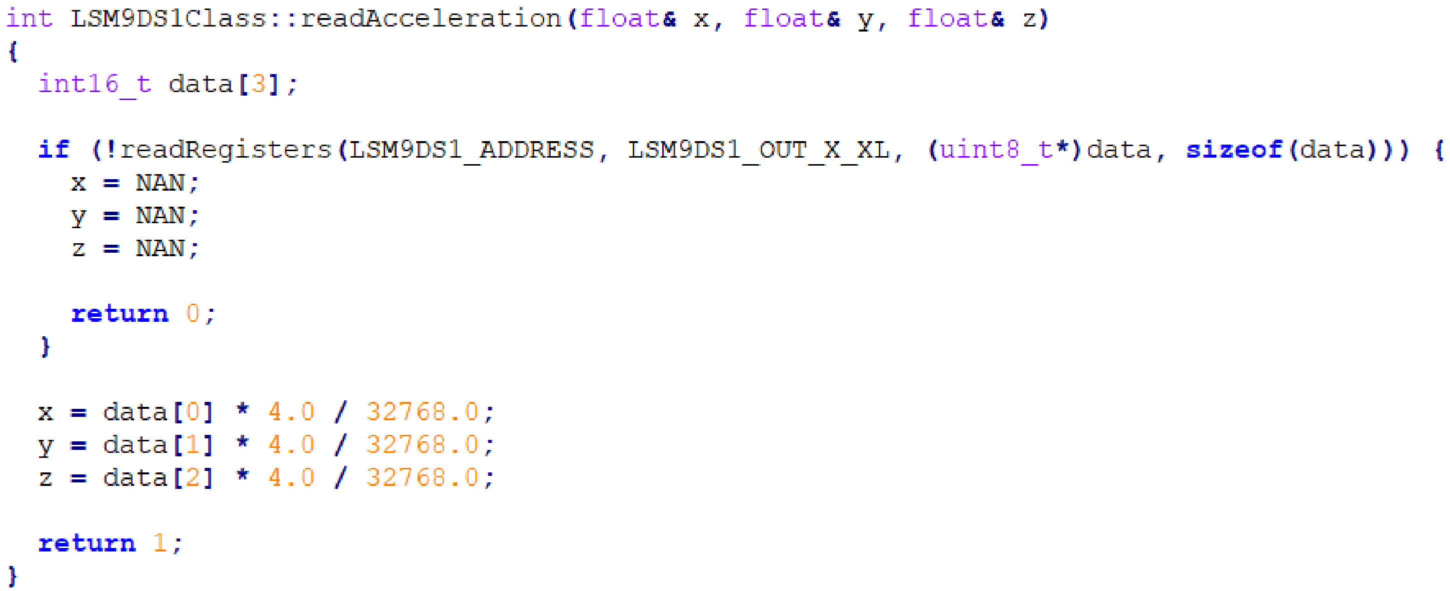
LSM9DS1.cpp (source code) – snap shots



```
#include "LSM9DS1.h"

#define LSM9DS1_ADDRESS            0x6b

#define LSM9DS1_WHO_AM_I           0x0f
#define LSM9DS1_CTRL_REG1_G        0x10
#define LSM9DS1_STATUS_REG         0x17
#define LSM9DS1_OUT_X_G            0x18
#define LSM9DS1_CTRL_REG6_XL       0x20
#define LSM9DS1_CTRL_REG8          0x22
#define LSM9DS1_OUT_X_XL           0x28
```



```
 writeRegister(LSM9DS1_ADDRESS, LSM9DS1_CTRL_REG1_G, 0x78); // 119 Hz, 2000 dps, 16 Hz BW
 writeRegister(LSM9DS1_ADDRESS, LSM9DS1_CTRL_REG6_XL, 0x70); // 119 Hz, 4G

 writeRegister(LSM9DS1_ADDRESS_M, LSM9DS1_CTRL_REG1_M, 0xb4); // Temperature compensation enable, medium performance, 20 Hz
 writeRegister(LSM9DS1_ADDRESS_M, LSM9DS1_CTRL_REG2_M, 0x00); // 4 Gauss
 writeRegister(LSM9DS1_ADDRESS_M, LSM9DS1_CTRL_REG3_M, 0x00); // Continuous conversion mode
```



```
int LSM9DS1Class::readAcceleration(float& x, float& y, float& z)
{
  int16_t data[3];

  if (!readRegisters(LSM9DS1_ADDRESS, LSM9DS1_OUT_X_XL, (uint8_t*)data, sizeof(data))) {
    x = NAN;
    y = NAN;
    z = NAN;

    return 0;
  }

  x = data[0] * 4.0 / 32768.0;
  y = data[1] * 4.0 / 32768.0;
  z = data[2] * 4.0 / 32768.0;

  return 1;
}
```

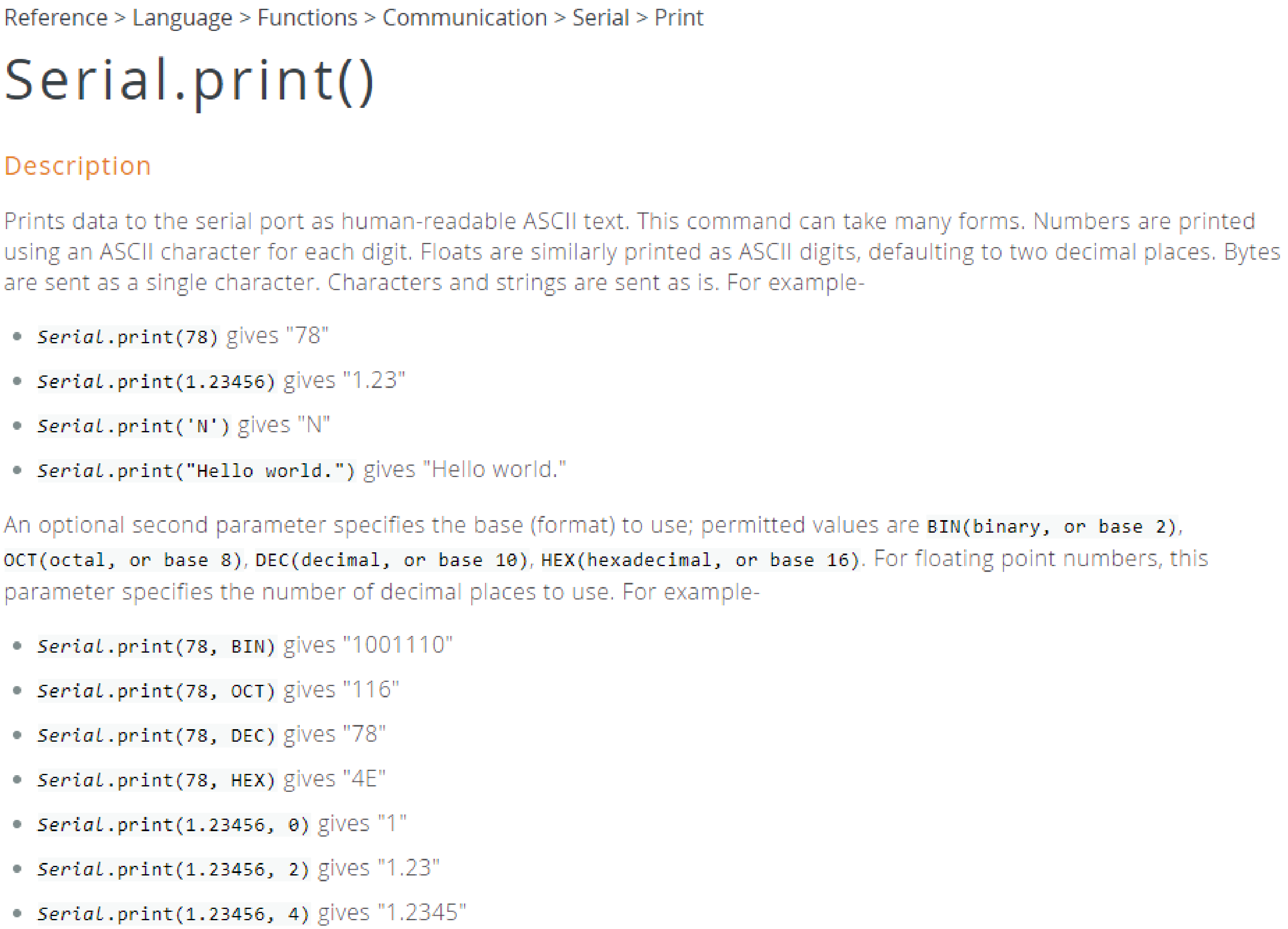
Interface between Arduino & LabVIEW

- LINX make it easy to interact with Arduino
  - Does not support the Nano 33

- **We will use standard serial communication**
  - Transmit/receive ASCII data
  - Transmit/receive binary data

LINX

LINX provides easy to use LabVIEW VIs for interacting with common embedded platforms like Arduino, chipKIT and myRIO. Use the built in sensor VIs to start getting data to your PC in seconds or use the peripheral VIs to access your devices digital I/O, analog I/O, SPI, I2C, UART, PWM and more.



Reference > Language > Functions > Communication > Serial > Print

Serial.print()

Description

Prints data to the serial port as human-readable ASCII text. This command can take many forms. Numbers are printed using an ASCII character for each digit. Floats are similarly printed as ASCII digits, defaulting to two decimal places. Bytes are sent as a single character. Characters and strings are sent as is. For example-

- `Serial.print(78)` gives "78"
- `Serial.print(1.23456)` gives "1.23"
- `Serial.print('N')` gives "N"
- `Serial.print("Hello world.")` gives "Hello world."

An optional second parameter specifies the base (format) to use; permitted values are `BIN(binary, or base 2)`, `OCT(octal, or base 8)`, `DEC(decimal, or base 10)`, `HEX(hexadecimal, or base 16)`. For floating point numbers, this parameter specifies the number of decimal places to use. For example-

- `Serial.print(78, BIN)` gives "1001110"
- `Serial.print(78, OCT)` gives "116"
- `Serial.print(78, DEC)` gives "78"
- `Serial.print(78, HEX)` gives "4E"
- `Serial.print(1.23456, 0)` gives "1"
- `Serial.print(1.23456, 2)` gives "1.23"
- `Serial.print(1.23456, 4)` gives "1.2345"

https://www.arduino.cc/reference/en/language/functions/communication/serial/print/

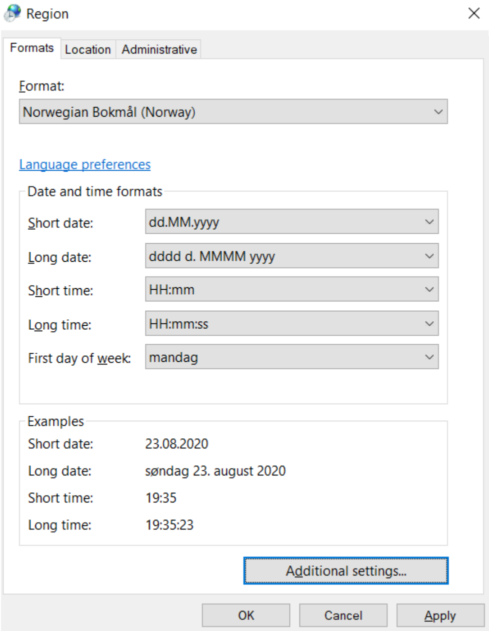
Regional settings

- Change decimal symbol to "."
  - Under ***Additional settings*** (see next slide for how to navigate there)

- Decimal symbol is standard "," in Norwgian, but "." in English.

- Sometimes decimal symbol "." is required!

- For instance the *Spreadsheet string to Array* function in LabVIEW is affected!
  - *Important for lab 3 and 4*



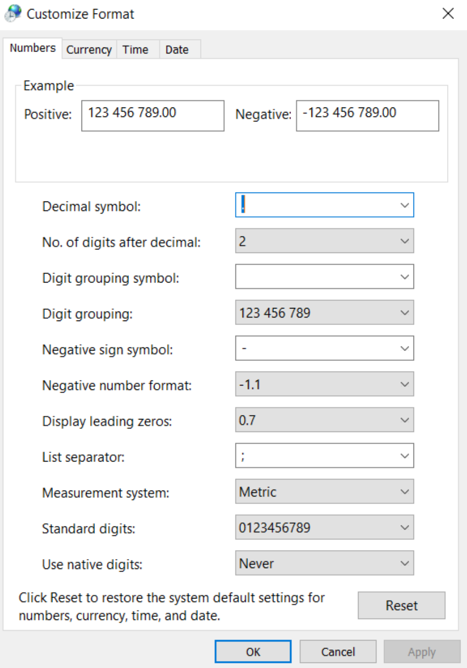
Format: Norwegian Bokmål (Norway)

Date and time formats

Short date:	dd.MM.yyyy
Long date:	dddd d. MMMM yyyy
Short time:	HH:mm
Long time:	HH:mm:ss
First day of week:	mandag

Examples

Short date:	23.08.2020
Long date:	søndag 23. august 2020
Short time:	19:35
Long time:	19:35:23



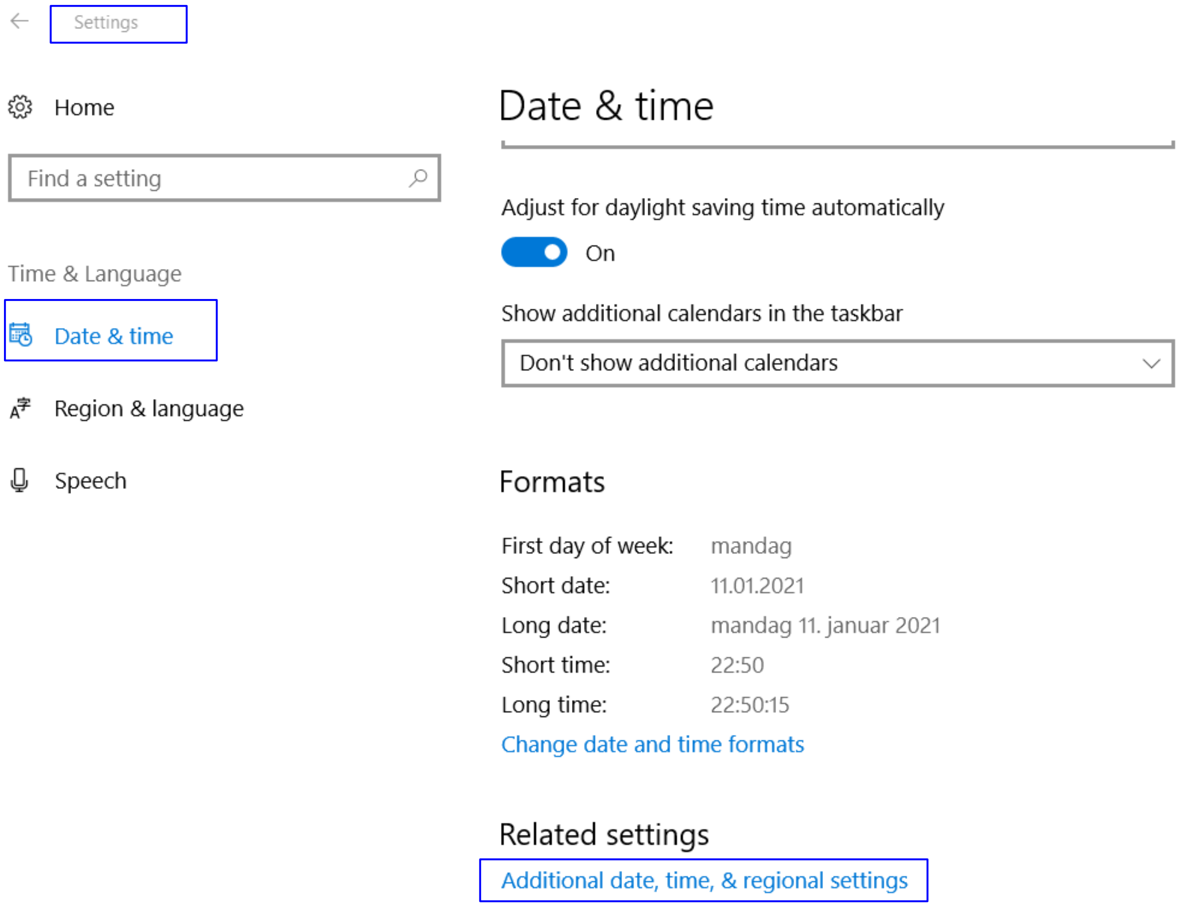
Example

Positive: 123 456 789.00 Negative: -123 456 789.00

Decimal symbol:	.
No. of digits after decimal:	2
Digit grouping symbol:	
Digit grouping:	123 456 789
Negative sign symbol:	-
Negative number format:	-1.1
Display leading zeros:	0.7
List separator:	;
Measurement system:	Metric
Standard digits:	0123456789
Use native digits:	Never

Click Reset to restore the system default settings for numbers, currency, time, and date.

Settings



Date & time

Adjust for daylight saving time automatically

On

Show additional calendars in the taskbar

Don't show additional calendars

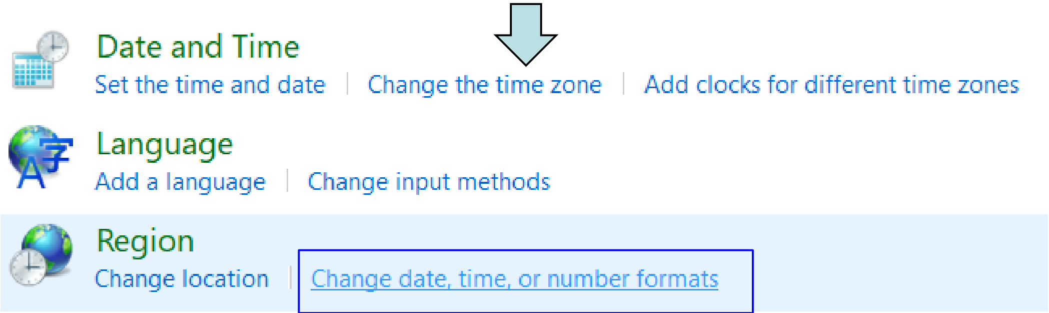
Formats

First day of week:	mandag
Short date:	11.01.2021
Long date:	mandag 11. januar 2021
Short time:	22:50
Long time:	22:50:15

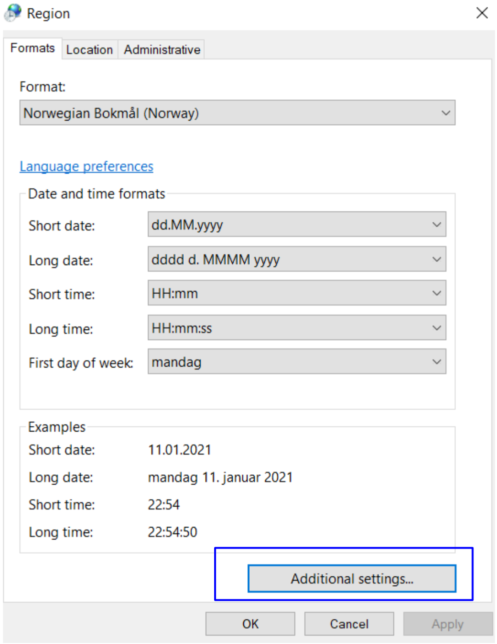
Change date and time formats

Related settings

Additional date, time, & regional settings



Date and Time
Set the time and date | Change the time zone | Add clocks for different time zones
Language
Add a language | Change input methods
Region
Change location | Change date, time, or number formats



Format: Norwegian Bokmål (Norway)

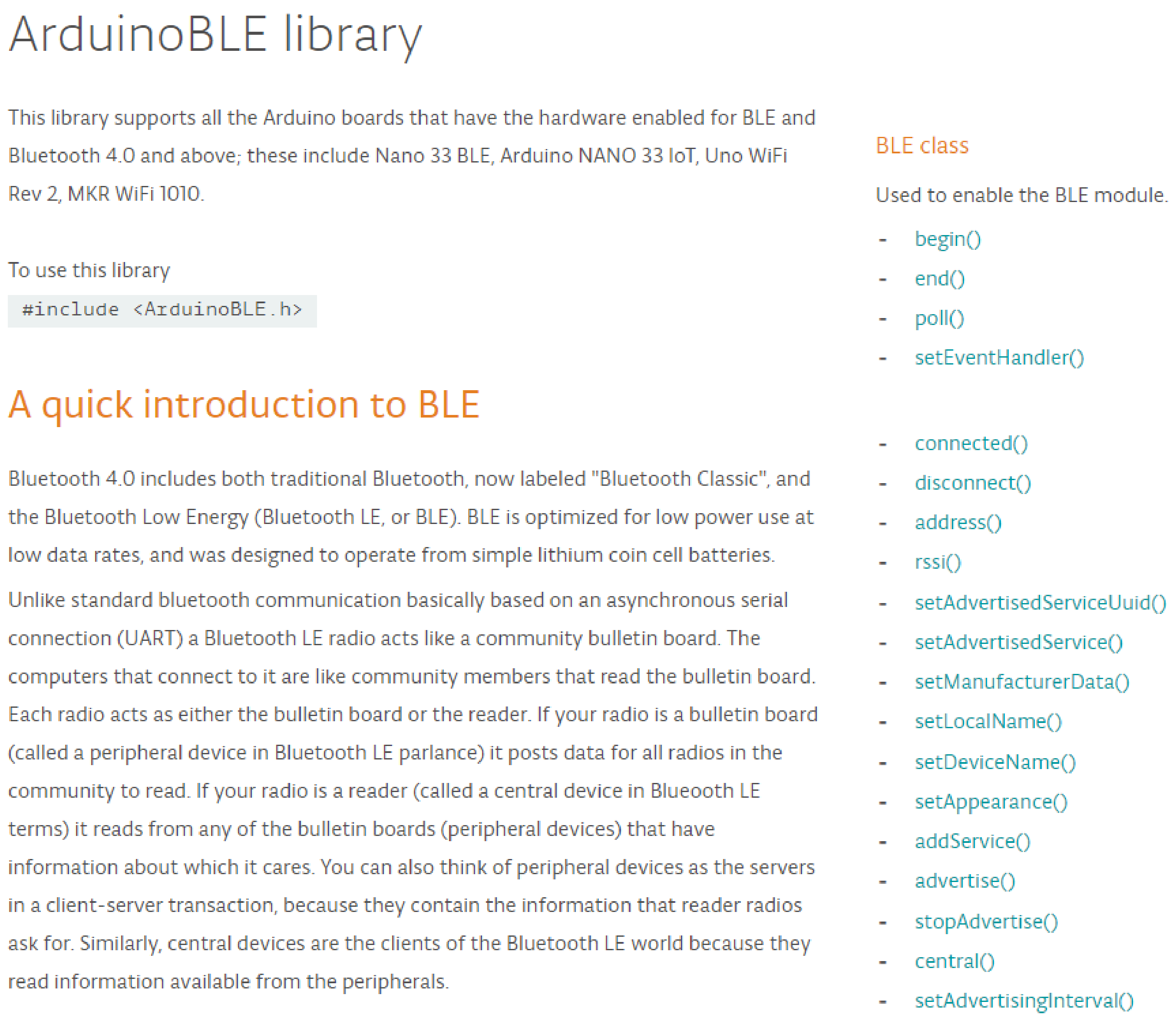
Language preferences

Date and time formats

Short date:	dd.MM.yyyy
Long date:	dddd d. MMMM yyyy
Short time:	HH:mm
Long time:	HH:mm:ss
First day of week:	mandag

Examples

Short date:	11.01.2021
Long date:	mandag 11. januar 2021
Short time:	22:54
Long time:	22:54:50



ArduinoBLE library

This library supports all the Arduino boards that have the hardware enabled for BLE and Bluetooth 4.0 and above; these include Nano 33 BLE, Arduino NANO 33 IoT, Uno WiFi Rev 2, MKR WiFi 1010.

To use this library

```
#include <ArduinoBLE.h>
```

A quick introduction to BLE

Bluetooth 4.0 includes both traditional Bluetooth, now labeled "Bluetooth Classic", and the Bluetooth Low Energy (Bluetooth LE, or BLE). BLE is optimized for low power use at low data rates, and was designed to operate from simple lithium coin cell batteries.

Unlike standard bluetooth communication basically based on an asynchronous serial connection (UART) a Bluetooth LE radio acts like a community bulletin board. The computers that connect to it are like community members that read the bulletin board. Each radio acts as either the bulletin board or the reader. If your radio is a bulletin board (called a peripheral device in Bluetooth LE parlance) it posts data for all radios in the community to read. If your radio is a reader (called a central device in Blueooth LE terms) it reads from any of the bulletin boards (peripheral devices) that have information about which it cares. You can also think of peripheral devices as the servers in a client-server transaction, because they contain the information that reader radios ask for. Similarly, central devices are the clients of the Bluetooth LE world because they read information available from the peripherals.

BLE class

Used to enable the BLE module.

- begin()
- end()
- poll()
- setEventHandler()

- connected()
- disconnect()
- address()
- rssi()
- setAdvertisedServiceUuid()
- setAdvertisedService()
- setManufacturerData()
- setLocalName()
- setDeviceName()
- setAppearance()
- addService()
- advertise()
- stopAdvertise()
- central()
- setAdvertisingInterval()

https://www.arduino.cc/en/Reference/ArduinoBLE

BLE examples

- https://rootsaid.com/arduino-ble-example/
- https://www.okdo.com/project/get-started-with-arduino-nano-33-sense/

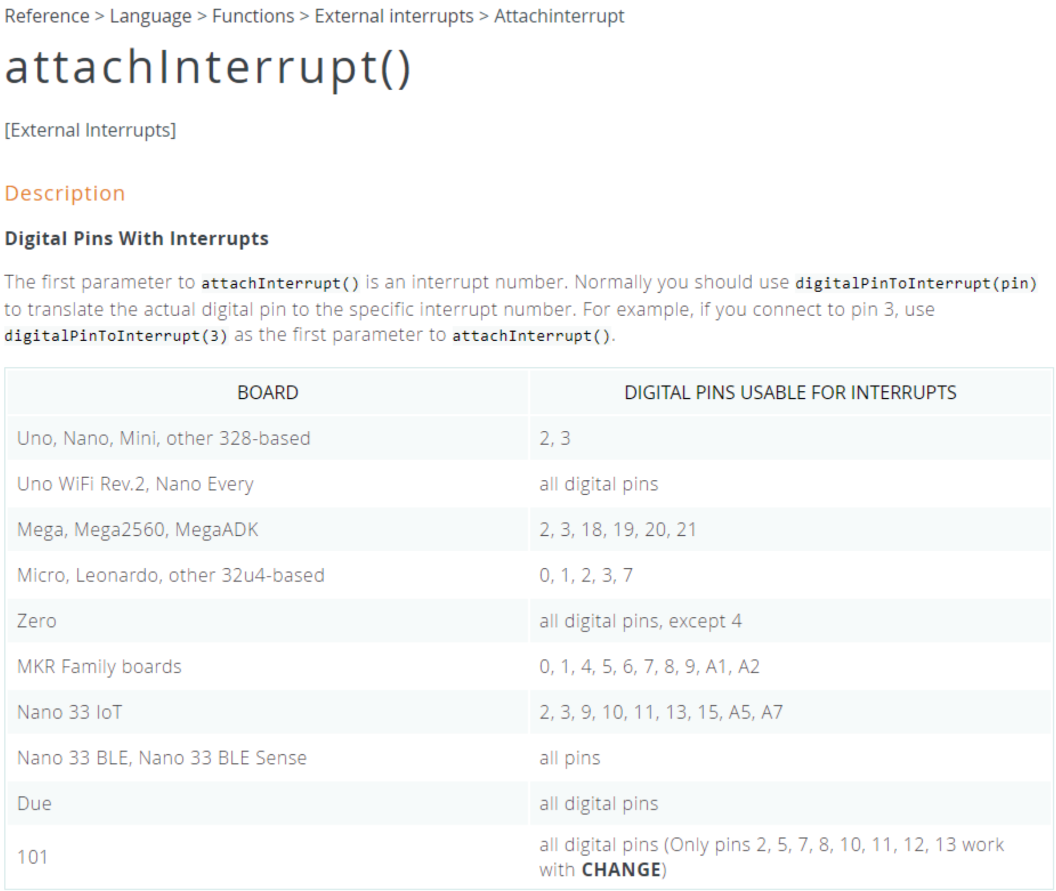
↖ And Nano 33 setup …

BLE connection to phones and computers

- To send/receive data or for control.
- Apps for test available for Android and IOS
- ArduinoBLE library does not support pairing (optional security feature in BLE)
  - ArduinoBLE can not connect to Windows 10 operating system.
  - Can use a BLE USB dongle and drivers (see lab 5)
- Can have range/stability issues …

Interrupts

https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/



Reference > Language > Functions > External interrupts > Attachinterrupt

attachInterrupt()

[External Interrupts]

Description

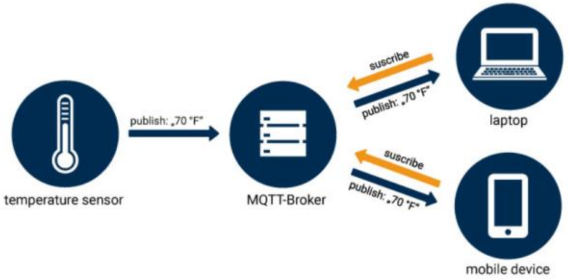
Digital Pins With Interrupts

The first parameter to `attachInterrupt()` is an interrupt number. Normally you should use `digitalPinToInterrupt(pin)` to translate the actual digital pin to the specific interrupt number. For example, if you connect to pin 3, use `digitalPinToInterrupt(3)` as the first parameter to `attachInterrupt()`.

BOARD	DIGITAL PINS USABLE FOR INTERRUPTS
Uno, Nano, Mini, other 328-based	2, 3
Uno WiFi Rev.2, Nano Every	all digital pins
Mega, Mega2560, MegaADK	2, 3, 18, 19, 20, 21
Micro, Leonardo, other 32u4-based	0, 1, 2, 3, 7
Zero	all digital pins, except 4
MKR Family boards	0, 1, 4, 5, 6, 7, 8, 9, A1, A2
Nano 33 IoT	2, 3, 9, 10, 11, 13, 15, A5, A7
Nano 33 BLE, Nano 33 BLE Sense	all pins
Due	all digital pins
101	all digital pins (Only pins 2, 5, 7, 8, 10, 11, 12, 13 work with **CHANGE**)

Common protocols for data transmission

- **JSON** (JavaScript Object Notation) - ASCII (text) based
  - name–value pair (key–value pair)
    - {'x' : 0.66, 'y' : 0.59, 'z' : -0.49}
  - See page 136-137 in course book
  - Supported in both Arduino and LabVIEW
  - Self Descriptive protocol
- BSON (binary protocol - binary version of JSON)
- **MQTT** (Message Queuing Telemetry Transport)
  - Publish subscribe
  - Common in IoT devices
  - (More info from page 607 in course book)

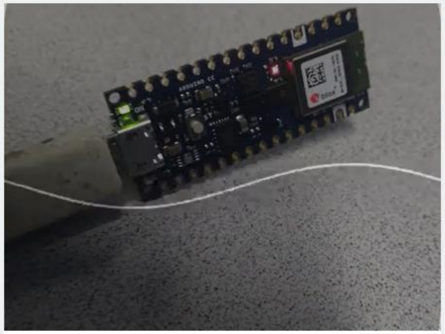


- Google Protocol buffers (binary protocol)

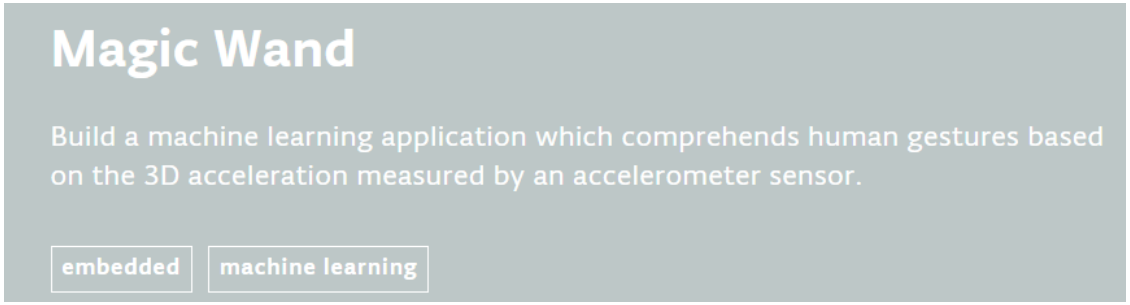
[What is MQTT? A practical introduction. (opc-router.com)](https://www.opc-router.com)

Nano 33 BLE Sense - Project Hub

https://create.arduino.cc/projecthub/products/nano-33-ble-sense



Wake Word Detection

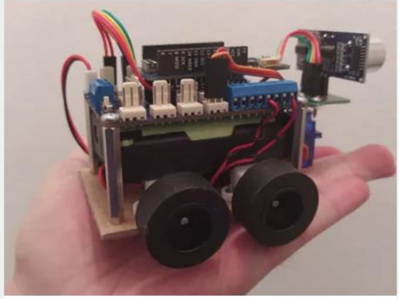


Magic Wand

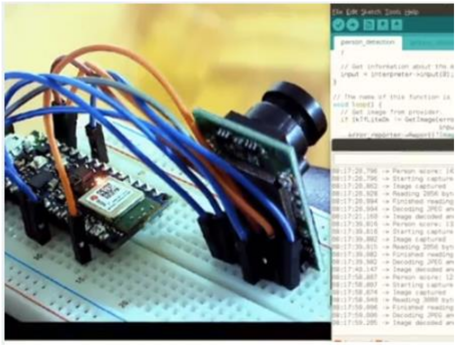
Build a machine learning application which comprehends human gestures based on the 3D acceleration measured by an accelerometer sensor.

embedded | machine learning

Arduino Nano 33 Sense | BLE Battery Level Tutorial



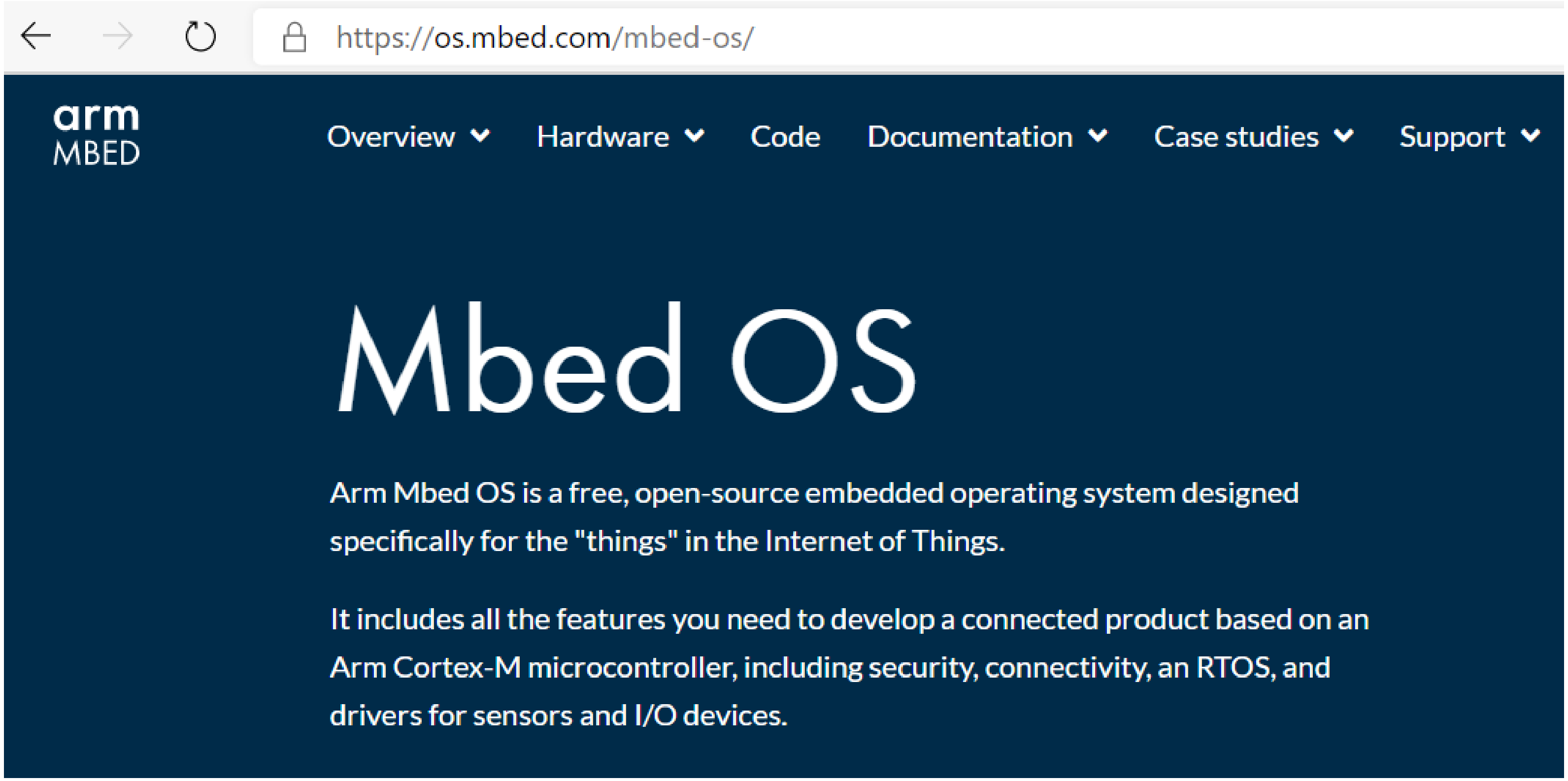
Mini 4WD Arduino Robot Controlled by Bluetooth



Person Detection with TensorFlow and Arduino

ARM Mbed OS

- Possible to run on Nano 33 BLE sense!
- We will not use it in this course



Mbed OS

Arm Mbed OS is a free, open-source embedded operating system designed specifically for the "things" in the Internet of Things.

It includes all the features you need to develop a connected product based on an Arm Cortex-M microcontroller, including security, connectivity, an RTOS, and drivers for sensors and I/O devices.