Table of Contents

ArduinoIDE 1
ArduinoIDE portable 1
ArduinoIDE with external Editor 1
ArduinoIDE terminal compile and program board 3
Finding the board name 4
ArduinoIDE modular code 4
ArduinoIDE Debug PRINT 6
ArduinoIDE alternative serial consoles 7
Meter código C en Arduino 7
Code Footprint 8

ArduinoIDE

arduino, iot, development, c programming

ArduinoIDE portable

- Download the distribution from the official webpage:

```
wget https://downloads.arduino.cc/arduino-1.8.13-linux64.tar.xz
tar xvf arduino-1.8.13-linux64.tar.xz
```

- Inside arduino-1.8.13/, create a directory named portable

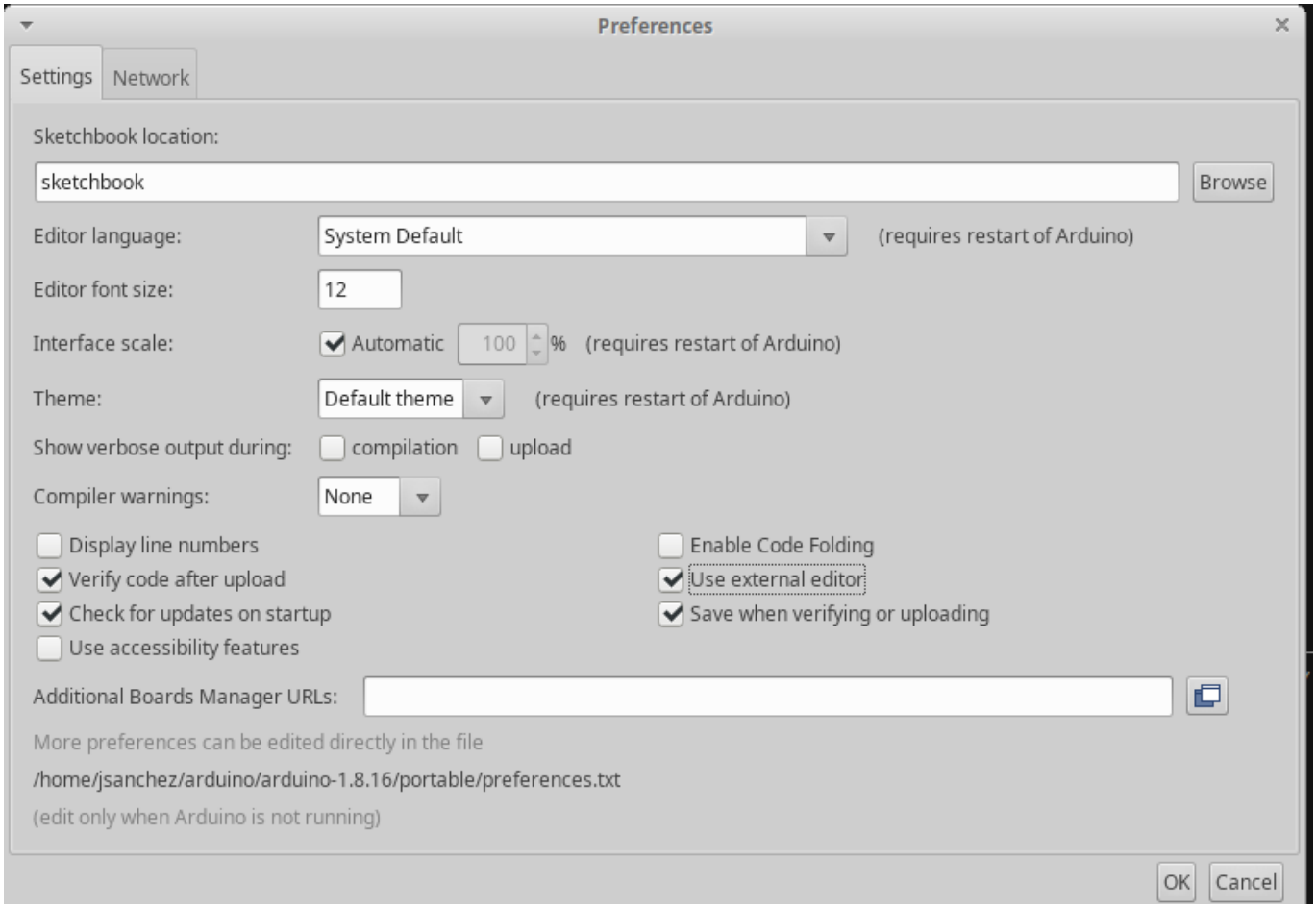
```
cd arduino-1.8.13/
mkdir portable
```

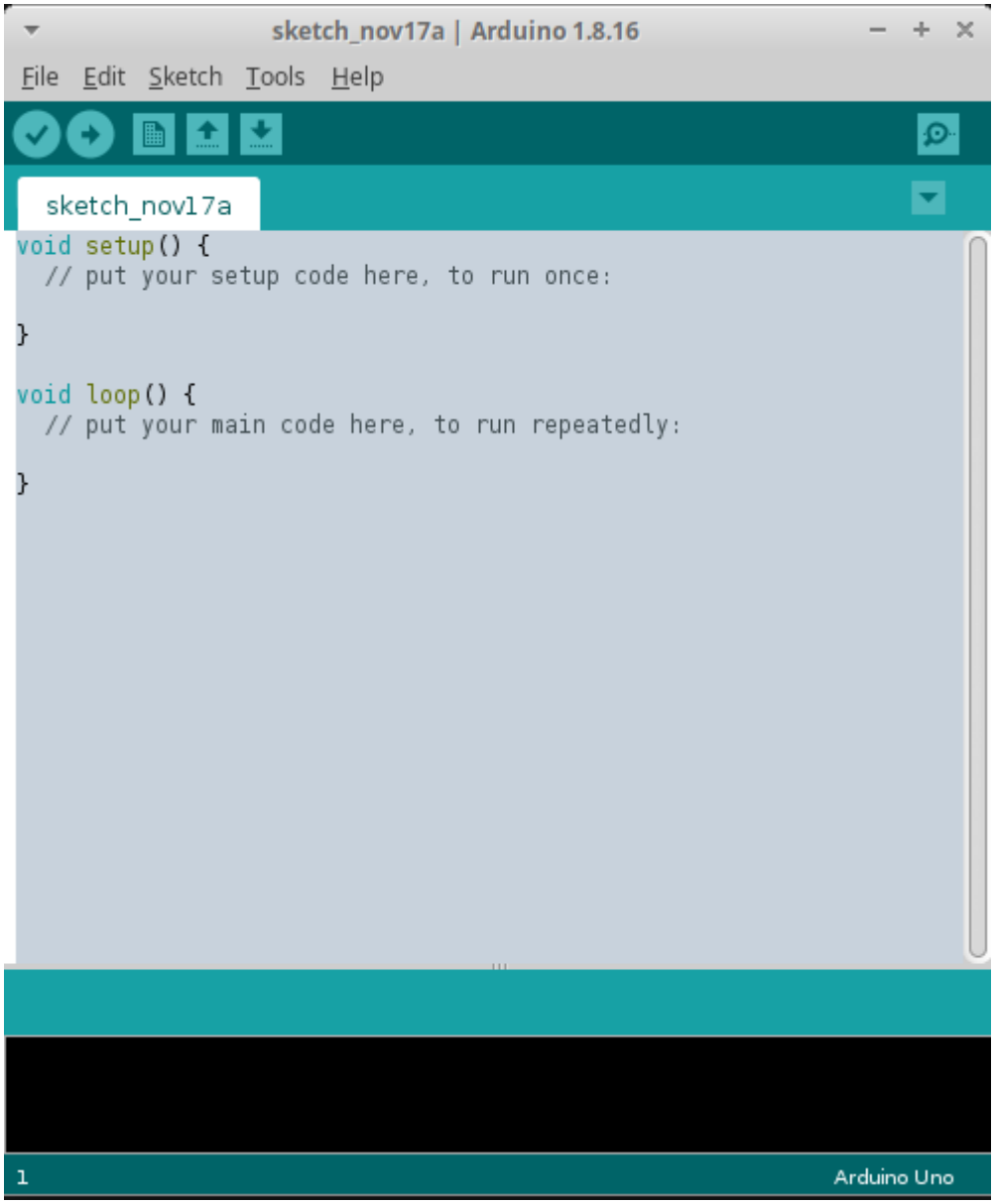
- By merely existing, this `portable` directory modiffies the base behaviour of the ArduinoIDE environment. From now on, all the libraries, compilers, utilities, and conffiguration files are always contained within the portable folder.

```
cd arduino-1.8.13/
./arduino
```

ArduinoIDE with external Editor

- Go to settings and mark the `Use External Editor` option.
- Now you can edit the sketch files with your favourite editor and save.
- When you're done with editing, use ArduinoIDE `Upload`and Serial Console normally.


Preferences
Settings
Network
Sketchbook location:
sketchbook
Browse
Editor language:
System Default
(requires restart of Arduino)
Editor font size:
12
Interface scale:
Automatic
100
%
(requires restart of Arduino)
Theme:
Default theme
(requires restart of Arduino)
Show verbose output during:
compilation
upload
Compiler warnings:
None
Display line numbers
Enable Code Folding
Verify code after upload
Use external editor
Check for updates on startup
Save when verifying or uploading
Use accessibility features
Additional Boards Manager URLs:
More preferences can be edited directly in the file
/home/jsanchez/arduino/arduino-1.8.16/portable/preferences.txt
(edit only when Arduino is not running)
OK
Cancel



ArduinoIDE terminal compile and program board

- You can compile and program the board using ArduinoIDE from the terminal:

```
./arduino
    --port /dev/ttyUSB0
    --board Arrow:samd:SmartEverything_Fox_native
    --preserve-temp-files --pref build.path=/home/jsanchez/tmp/arduinobuild
    --verify || OR || --upload
    ./MYSKETCH.ino
```

- `--port` must be set to your serial device port.
- `--board` must be set to your board model (more on this next).
- `--preserve-temp-files` tells the compiler to build only the updated files instead of all the

code.
    - This is a HUGE time saver!.
    - Must specify with –pref build.path the directory to contain all the temporary object files. This directory can be safely be deleted later.
- `--verify` only compiles the code without uploading it to the board (nice for debugging).
- `--upload` compiles and uploads the code to the board.

Don't forget the `./` in front of your sketch name!!

Finding the board name

```
./arduino
  --port /dev/ttyUSB0
  --board Arrow:samd:SmartEverything_Fox_native
  --preserve-temp-files --pref build.path=/home/jsanchez/tmp/arduinobuild
  --verify || OR || --upload
  ./MYSKETCH.ino
```

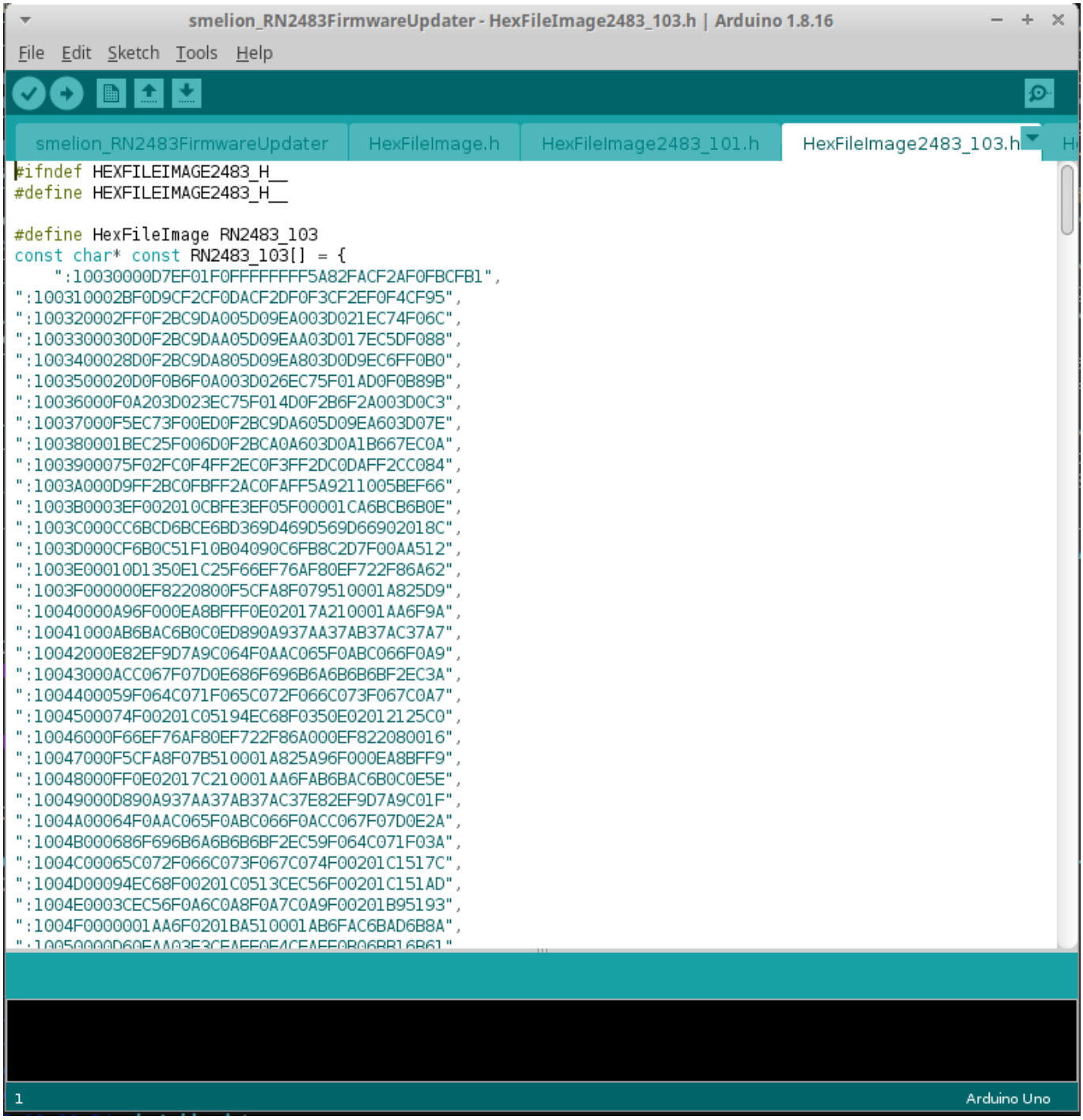
- `--board` must be set to your board model.
- You can find several boards.txt files in your source tree.

```
hardware/arduino/avr/boards.txt
portable/packages/Arrow/hardware/samd/2.1.0/boards.txt
portable/packages/arduino/hardware/samd/1.6.18/boards.txt
```

You can build the board following the dir names and within the boards.txt file itself:
`SmartEverything_Fox_native.name=SmartEverything Fox (Native USB Port)`

ArduinoIDE modular code

- Besides your `.ino` file, you can place additional source files within your sketch folder.
- These will be compiled and linked implicitly by the ArduinoIDE toolchain.
- However, don't forget to do apply the correct C/C++ modular code practices!! i.e. `#include`, `extern`, etc.
- Note how the files are listed as TABS in the ArduinoIDE interface.


smelion_RN2483FirmwareUpdater - HexFileImage2483_103.h | Arduino 1.8.16
File Edit Sketch Tools Help
smelion_RN2483FirmwareUpdater
HexFileImage.h
HexFileImage2483_101.h
HexFileImage2483_103.h
#ifndef HEXFILEIMAGE2483_H__
#define HEXFILEIMAGE2483_H__
#define HexFileImage RN2483_103
const char* const RN2483_103[] = {
":10030000D7EF01F0FFFFFFFF5A82FACF2AF0FBCFB1",
":100310002BF0D9CF2CF0DACF2DF0F3CF2EF0F4CF95",
":100320002FF0F2BC9DA005D09EA003D021EC74F06C",
":1003300030D0F2BC9DAA05D09EAA03D017EC5DF088",
":1003400028D0F2BC9DA805D09EA803D0D9EC6FF0B0",
":1003500020D0F0B6F0A003D026EC75F01AD0F0B89B",
":10036000F0A203D023EC75F014D0F2B6F2A003D0C3",
":10037000F5EC73F00ED0F2BC9DA605D09EA603D07E",
":100380001BEC25F006D0F2BCA0A603D0A1B667EC0A",
":1003900075F02FC0F4FF2EC0F3FF2DC0DAFF2CC084",
":1003A000D9FF2BC0FBFF2AC0FAFF5A9211005BEF66",
":1003B0003EF002010CBFE3EF05F00001CA6BCB6B0E",
":1003C000CC6BCD6BCE6BD369D469D569D669020180",
":1003D000CF6B0C51F10B04090C6FB8C2D7F00AA512",
":1003E00010D1350E1C25F66EF76AF80EF722F86A62",
":1003F000000EF8220800F5CFA8F079510001A825D9",
":10040000A96F000EA8BFFF0E02017A210001AA6F9A",
":10041000AB6BAC6B0C0ED890A937AA37AB37AC37A7",
":10042000E82EF9D7A9C064F0AAC065F0ABC066F0A9",
":10043000ACC067F07D0E686F696B6A6B6B6BF2EC3A",
":1004400059F064C071F065C072F066C073F067C0A7",
":1004500074F00201C05194EC68F0350E02012125C0",
":10046000F66EF76AF80EF722F86A000EF822080016",
":10047000F5CFA8F07B510001A825A96F000EA8BFF9",
":10048000FF0E02017C210001AA6FAB6BAC6B0C0E5E",
":10049000D890A937AA37AB37AC37E82EF9D7A9C01F",
":1004A00064F0AAC065F0ABC066F0ACC067F07D0E2A",
":1004B000686F696B6A6B6B6BF2EC59F064C071F03A",
":1004C00065C072F066C073F067C074F00201C1517C",
":1004D00094EC68F00201C0513CEC56F00201C151AD",
":1004E0003CEC56F0A6C0A8F0A7C0A9F00201B95193",
":1004F0000001AA6F0201BA510001AB6FAC6BAD6B8A",
1
Arduino Uno

```
[puesto8:smelion_RN2483FirmwareUpdater] jsanchez git:(8236dc4) ✗
$ ll
.rw-rw-r--  828 jsanchez jsanchez 2021-07-08 12:50 HexFileImage.h
.rw-rw-r-- 206k jsanchez jsanchez 2021-07-08 12:50 HexFileImage2483_101.h
.rw-rw-r-- 190k jsanchez jsanchez 2021-07-08 12:50 HexFileImage2483_103.h
.rw-rw-r-- 190k jsanchez jsanchez 2021-07-08 12:50 HexFileImage2483_106_RC3.h
.rw-rw-r-- 190k jsanchez jsanchez 2021-07-08 12:50 HexFileImage2903_098.h
.rw-rw-r-- 206k jsanchez jsanchez 2021-07-08 12:50 HexFileImage2903AU_097rc7.h
.rw-rw-r-- 9,3k jsanchez jsanchez 2021-07-08 12:50 IntelHexParser.cpp
.rw-rw-r-- 1,5k jsanchez jsanchez 2021-07-08 12:50 IntelHexParser.h
.rw-rw-r--  11k jsanchez jsanchez 2021-09-05 17:07 Readme.md
.rw-rw-r-- 7,8k jsanchez jsanchez 2021-07-08 12:50 RN2483Bootloader.cpp
.rw-rw-r-- 2,9k jsanchez jsanchez 2021-07-08 12:50 RN2483Bootloader.h
.rw-rw-r-- 8,5k jsanchez jsanchez 2021-09-05 17:07 smelion_RN2483FirmwareUpdater.ino
.rw-rw-r-- 5,2k jsanchez jsanchez 2021-07-08 12:50 Sodaq_wdt.cpp
.rw-rw-r-- 1,8k jsanchez jsanchez 2021-07-08 12:50 Sodaq_wdt.h
.rw-rw-r-- 1,2k jsanchez jsanchez 2021-07-08 12:50 Utils.h
```

ArduinoIDE Debug PRINT

- Print function calls can be controlled across a whole `.c` file without rewriting code.
- This enables us to switch the debug PRINT on/off with a simple macro definition `#define DEBUG`.
- This is implemented in some way or another in difierent projects. It is a very widespread practice.
- Copy this block in your C file.

```
#ifdef DEBUG

  #define PRINT(...) Serial.print(__VA_ARGS__)
  #define PRINTLN(...) Serial.println(__VA_ARGS__)
  #define PRINT_ARRAY(add, len) \
  do { \
    int i; \
    for (i = 0 ; i < (len) ; i++) { \
      Serial.print((unsigned int)((uint8_t*)(add))[i], HEX); \
    } \
    Serial.println(); \
  } while(0)

#else /* DEBUG */

  #define PRINT(...)
  #define PRINTLN(...)
  #define PRINT_ARRAY(add, len)

#endif /* DEBUG */
```

- Then, write conditional print sentences as `PRINT(var)`.
- Additionally, the `PRINT ARRAY(address, len)` function simply prints in HEX an array passed

as an argument.
- **NOTE** to activate the conditional debut pring, `#define DEBUG` must be writen **before** the previous code block.
    - Alternatively, it can be defined with a compiler CFLAG environment variable.

      Use example:

```
uint8_t foo[255];
int bar = 7;
...
PRINTLN(bar);
PRINT_ARRAY(foo, sizeof(foo))
// These prints only if the DEBUG macro was defined.
7
EE FF 01 25 ...
```

ArduinoIDE alternative serial consoles

- ArduinoIDE comes with an embedded serial console.
- Alternatively you can use a serial console like picocom, but you must set it to the right parameters.

```
picocom

  -g "logs/serial_console_log.txt" # Save the consolo text in a log file
  -r                               # NO-reset - avoid reseting the device
  -b 115200                        # Baudrate - MUST MATCH ARDUINOIDE!
  --omap crcrlf                    # Mapping of EOL characters
  /dev/ttyACM0
```

- Exit Picocom with `Ctrl+A`, `Ctrl+X`.

Meter código C en Arduino

Se pueden meter archivos `.c` y sus respectivos `.h` en el mismo directorio que el sketch:

Fuente: https://forum.arduino.cc/index.php?topic=45003.0

C++, which the Arduino is programmed in, performs name-mangling. C, which your external functions are written in, does not.

Name-mangling means that the actual function that gets called is called using a name that is composed of the class name, function name, and argument type names.

In order to call a C function from a C++ function, the compiler needs to know that it should use C calling syntax, not C++ calling syntax.

In the header file, before any function declarations, add:

```
#ifdef __cplusplus
extern "C" {
#endif
```

After all function declarations, add:

```
#ifdef __cplusplus
}
#endif
```

This will allow the C functions to be called from C++.

Code Footprint

Reducir el tamaño de los firmwares

Por defecto, arduino le pasa al compilador el Flag `-g`, este flat añade a los archivos `.o` varios campos de depuración, que son listados en el `.map` como `.debug_info`, `.debug_range`, etc. Este código incrementa considerablemente el tamaño del firmware.

Para eliminar el código de depuración de los archivos objeto, editar el archivo `platform.txt` y quitar la opción `-g` de todos los flags de compilación.

- Los flags de compilación se encuentran en el archivo `platform.txt`, son líneas con el nombre `compiler`.

  E.g.:

```
arduino-1.8.0/portable/packages/Arrow/hardware/samd/2.1.0/platform.txt
```

Luego compilar los firmwares respectivos, abrir el directorio de compilación y utilizar el script:

```
python3 ~/bin/analyze_map.py coap_eap_integration.ino.map
```

Los archivos que pertenecen a cada una de las partes del código están listados en el readme de cada firmware.

From:
https://wiki.odins.es/ - **OdinS Wiki**

Permanent link:
https://wiki.odins.es/public/development/arduinoide?rev=1655898384

Last update: **2024/10/09 08:35**