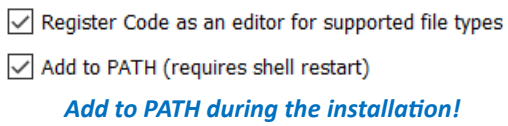
USB Custom Human Interface Device + RUST Tutorial

The document shows how to build your own USB HID device from scratch, which also calls RUST code from C, on an STM32F412ZG microcontroller. I have tried to present it in a flexible way that can be implemented on other devices. You can find the resources in one at the bottom. Good luck!

Bence Kristóf Szabó

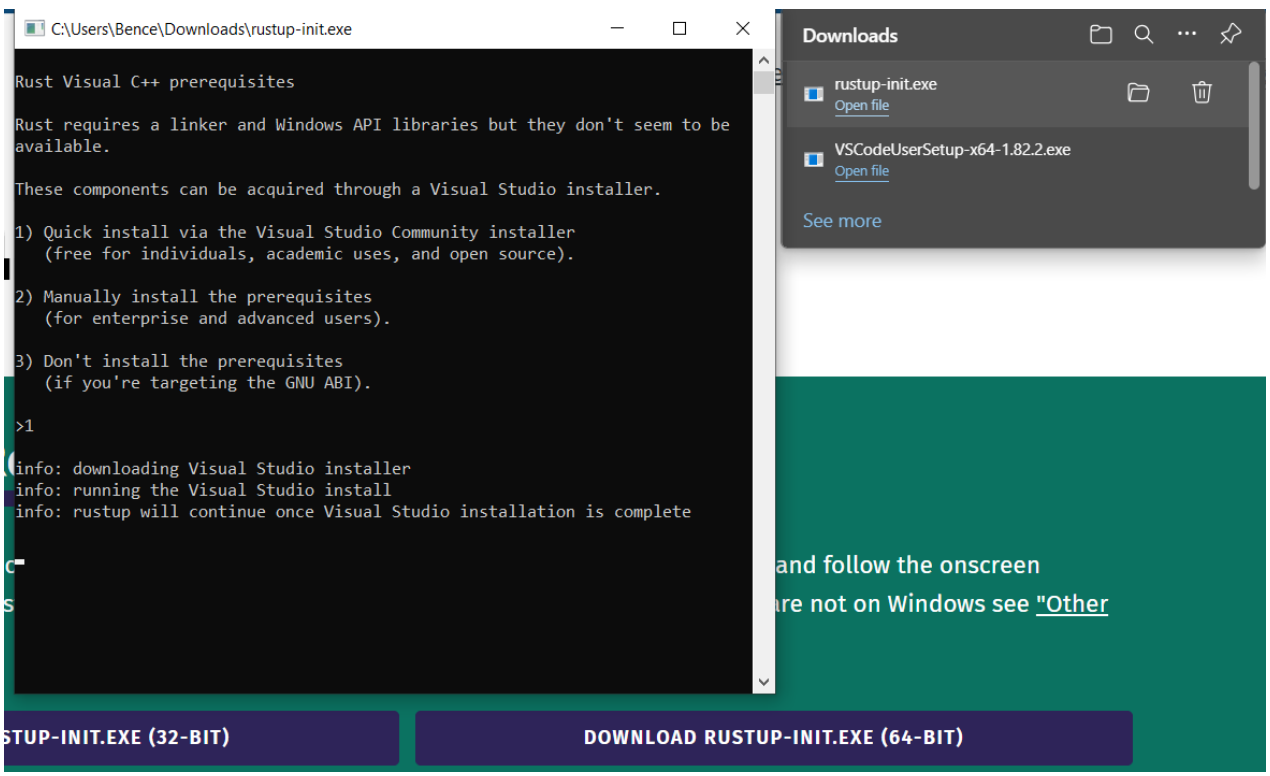
1. Creating the environment

First step: Download Visual Studio Code: https://code.visualstudio.com/download

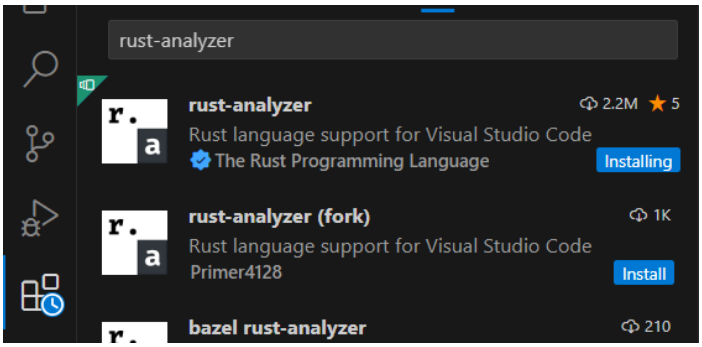


Next to get the RUST installer -> https://www.rust-lang.org/tools/install

After running, enter the number of the desired installation option at the command line: 1 - 2 – 3



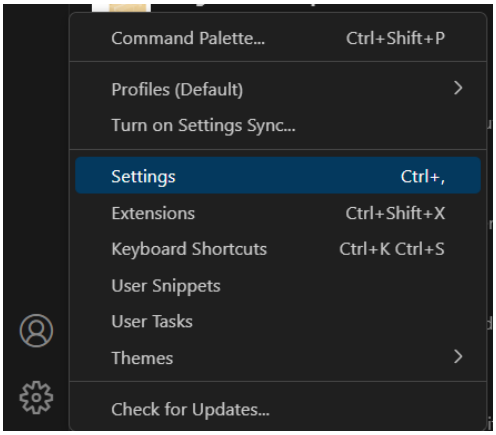
Once the rustup has uploaded itself, choose again from the options! When it is ready, in VSCode, install the extension called Rust-analyzer:

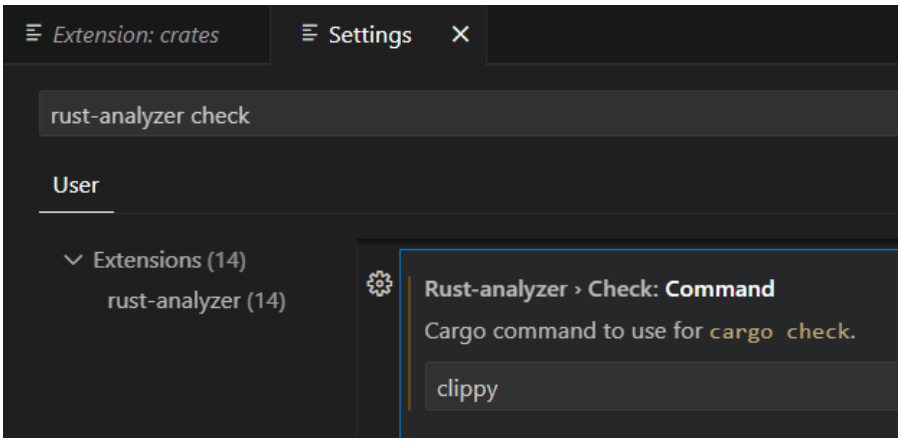


You can also follow the next part by watching Youtube videos (at the bottom of this page).
Comments like this are marked with **.

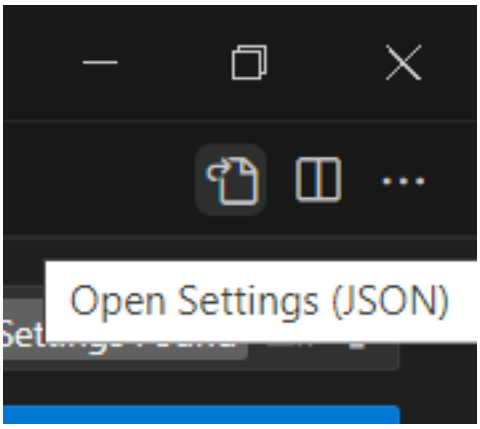
More useful extensions: crates, CodeLLDB, Even Better TOML, Error Lens.

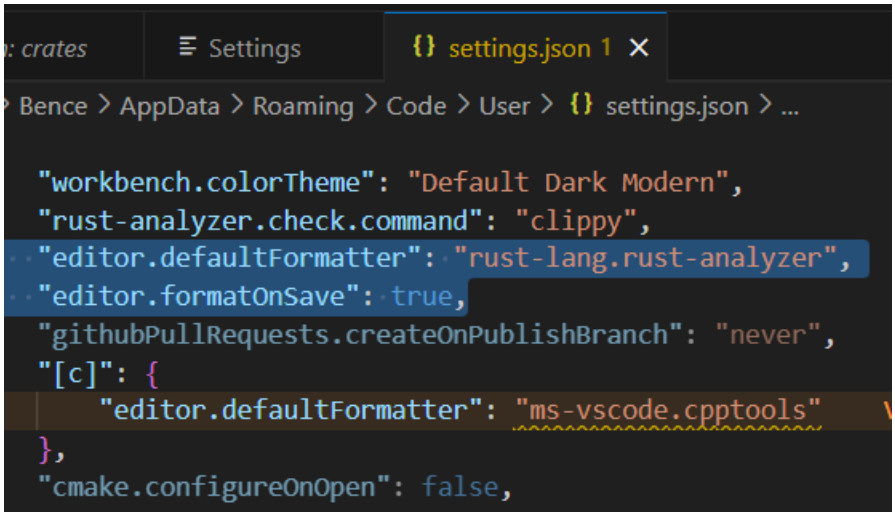
For more accurate feedback, change the “rust-analyzer check” to “clippy”, if it is not already.



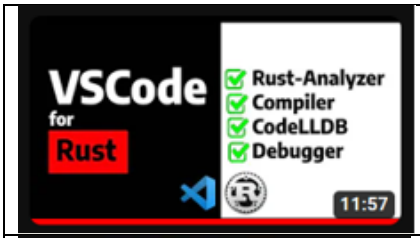


Click on "Open Settings (JSON)" and check the following:





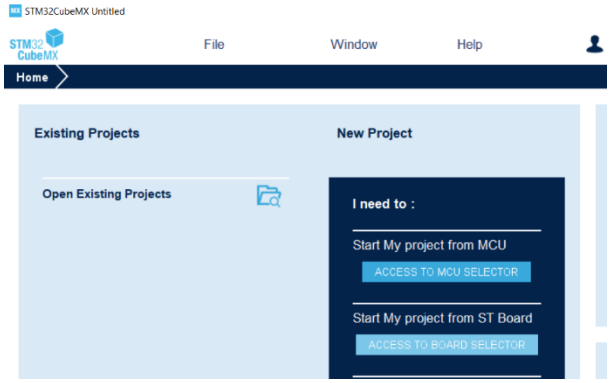
** Here are also 2 great videos about RUST installation and extensions.

	https://www.youtube.com/watch?v=yo4kWLtSPCY
	https://www.youtube.com/watch?v=BU1LYFkpJuk

2. Generate Custom HID with CubeMX

Download and install CubeMX: https://www.st.com/en/development-tools/stm32cubemx.html

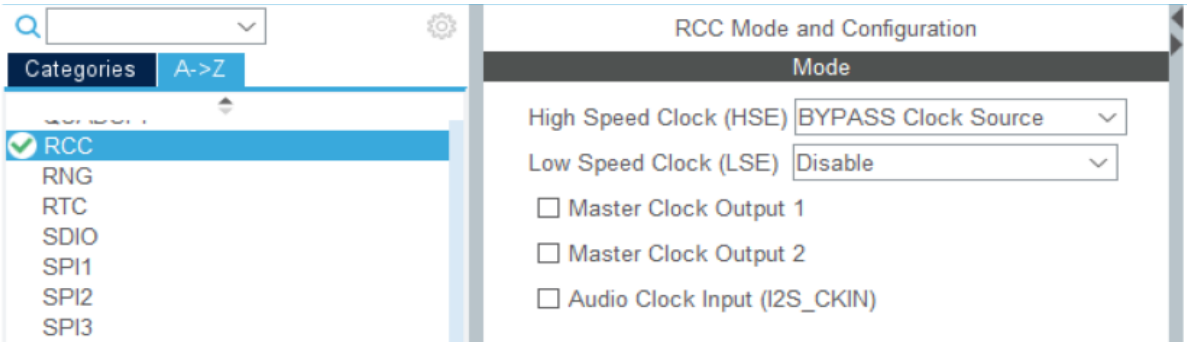
Create a Custom_USB_HID:



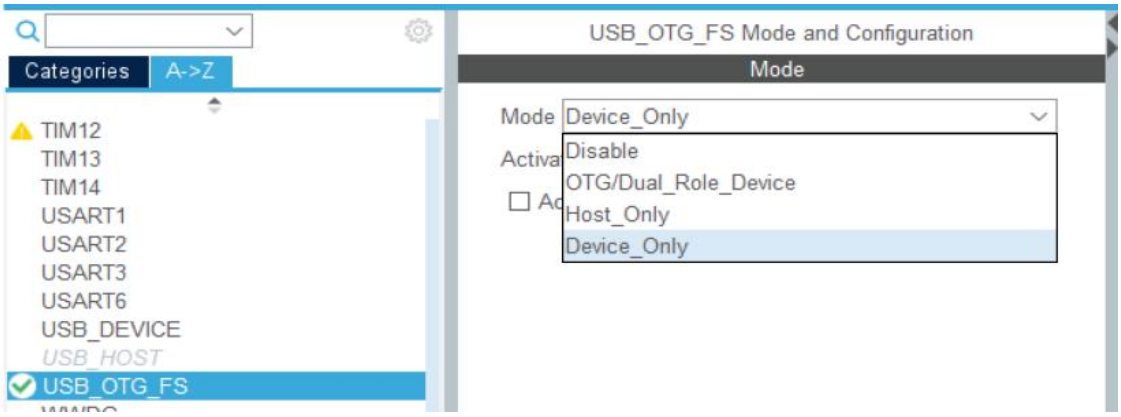
First, specify which pins you want to use.
(** Leds: PB0, PB7, PB14, UserButton: PC13 in - STM32F412ZG – case I remember these).

1. Access to board selector (Choose your mcu)
2. Clear Pinouts (Delete default settings!)
3. Setup Output/Input pin(s) (LEDs, Button), to set own label >> right-click, "Enter user label".

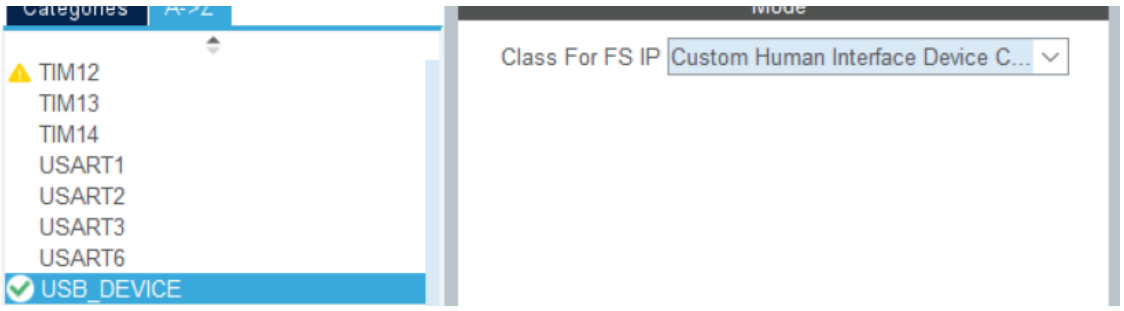
Now let's set the ones that make our microcontroller a custom usb device.
- First what we need is to set RCC, High Speed Clock >> BYPASS Clock Source



- Next one is to set USB_OTG_FS >> Device_Only

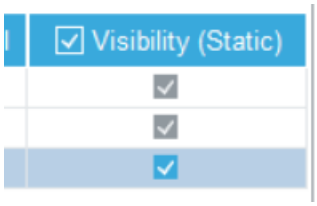


- Then set USB_DEVICE >> Custom Human Interface Device Class



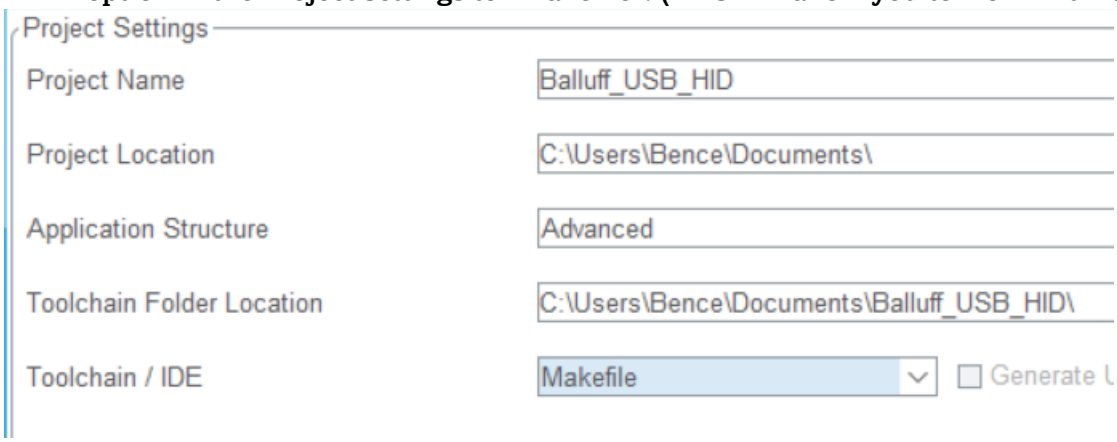
- Also here at USB_DEVICE, just further down, you can rewrite PRODUCT_STRING. (You can name your device)

- Above the big blue tabs, switch to >> "Clock Configuration" and if it offers to accept the automatic settings, if you are not prepared to set the values yourself.

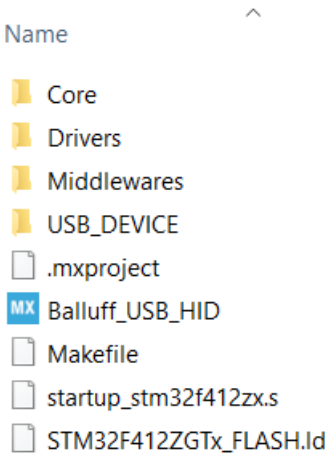


Next big blue tab >> "Project Manager" within that Advanced settings >> Visibility should all be ticked. Click on the blue bar at the top.

In the "Project Manager", the most important thing to do now is to change the "Toolchain / IDE" option in the Project settings to "Makefile". (This will allow you to work with it in VSCode)



- If you need something (e.g. your microcontroller files), CubeMX will throw it up and you can download it. It may ask you to log in, etc...etc. Do it and you will be rewarded with a successfully generated code.



** If something fails, the videos below will help you to do it, but you should do it in cubeMX, not CubeIDE.
https://www.youtube.com/watch?v=3JGRt3BFYrM
Don't forget, you will need "Makefile" https://www.youtube.com/watch?v=ZP2fd2qatj0

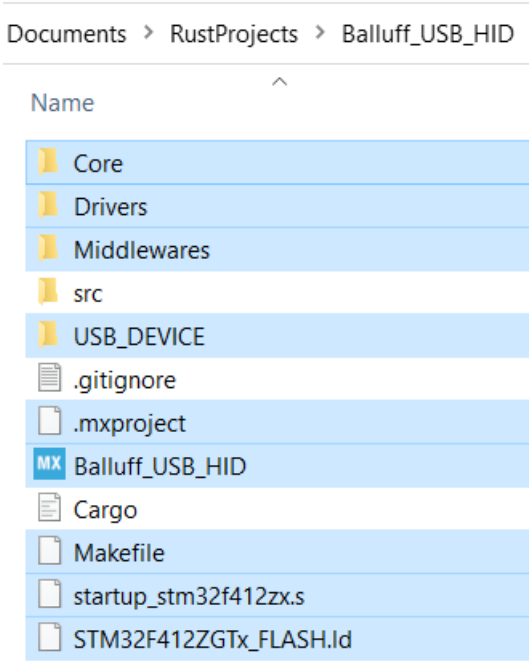
3. Creating the RUST project

In VSCode, create a RUST project with the same name as your cubeMX project name.

$ cargo new [name of your project] --lib (**If you want an app at any other time --bin)

```
PS C:\Users\Bence> cd C:\Users\Bence\Documents\RustProjects
PS C:\Users\Bence\Documents\RustProjects> cargo new Balluff_USB_HID --lib
     Created library `Balluff_USB_HID` package
PS C:\Users\Bence\Documents\RustProjects>
```

Copy the generated files to the folder of this RUST project!

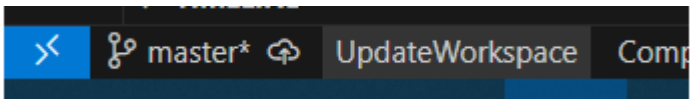


Open the folder and click on the file "src/lib.rs".
Once rust-analyzer has run, install the following extensions:

- C/C++
- C/C++ Extension Pack
- Makefile Tools
- STM32 VS Code Extension
- STM-Helper.

(**Don't be surprised to see several extensions installed with these)

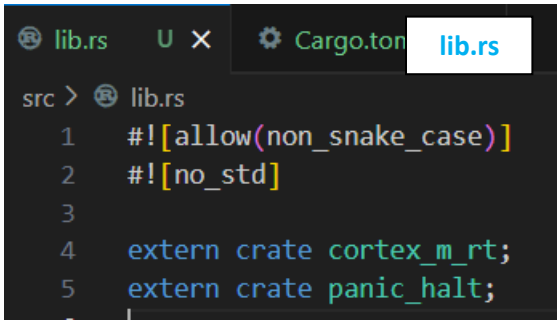
Thanks to the helper, new options will appear below, click on "UpdateWorkspace"!

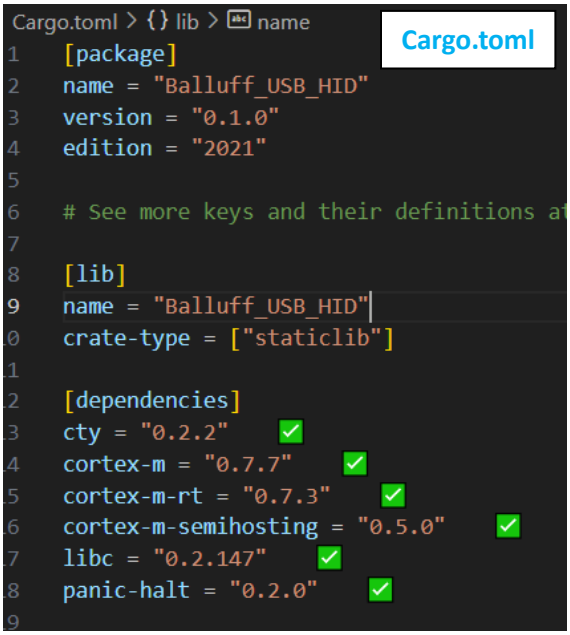


Generated a folder named ".vscode". Press "build"! If everything was done correctly, it ran successfully.
**I got an error in the file "c_cpp_properties" because it could not find a folder. The path had version 9.2.1, but my machine has 10.3.1. Solution: just had to rewrite the numbers in the path.

Modify the "cargo.toml" and "src/lib.rs" files

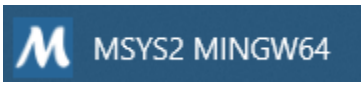
as follows:





Download MSYS2 so that both the C and RUST files use the same gcc builder: https://www.msys2.org/

On the page, in point 6, you can see that once you have finished the installation, you should run:
pacman -S mingw-w64-ucrt-x86_64-gcc command.



Start MSYS2 MINGW64 and type:
pacman -S mingw-w64-x86_64-rust

**Essentially, the same crates and cargoes need to be downloaded as when installing RUST.
Although (!) when writing the test program, VSCode also compiled the rust files correctly, whereas it did not in the original demo.

In VSCode terminal, run the command "rustup target add thumbv7em-none-eabihf".
For other microcontrollers you can check the target here:
https://docs.rust-embedded.org/book/intro/install.html

Use the "rustup target list" command to check if it is in the targeted (installed) state.

Place msys2/mingw64/bin at the top of your environment variables (Path).
".cargo/bin" should be the second one.

MSYS2 MinGW64 Terminal can also use "rustup target add thumbv7em-none-eabihf"

**If you complain that rustup is not a -bash command. Copy the rustup application from the folder "C:\Users\Users\Users\.cargo\bin" to the folder "(Path where you installed msyst)\msys2\mingw64\bin".
If you still fail to download the target, it may not be a problem. Because the program I created in parallel with this document, worked with cargo also.

1. Follow the instructions in the embedded book: https://docs.rust-embedded.org/book/intro/install/windows.html

2. Download the GNU Toolchain: https://developer.arm.com/downloads/-/gnu-rm

3/A. And OpenOCD also: https://github.com/xpack-dev-tools/openocd-xpack/releases/

3/B. Put the extracted content in a folder and add the bin directory to the PATH.

4. Install it from https://www.st.com/en/development-tools/stsw-link009.html, or OpenOCD will not work.

5. To test, type "openocd -v" in the terminal

4. Combining C and RUST

Now we're going to adjust some things according to this video so that the device works correctly: https://www.youtube.com/watch?v=3JGRt3BFYrM

Folder: USB_DEVICE >> Target >> usbd_conf.h in the USBD_CUSTOMHID_OUTREPORT_BUF_SIZE file should be set from "2U" to "64U", USBD_CUSTOM_HID_REPORT_DESC_SIZE should be set to "33U".

usbd_conf.h

```
/*---------- -----------*/
#define USBD_CUSTOMHID_OUTREPORT_BUF_SIZE      64U
/*---------- -----------*/
#define USBD_CUSTOM_HID_REPORT_DESC_SIZE       33U
/*---------- -----------*/
```

Folder: Middlewares\ST\STM32_USB_Device_Library\Class\CustomHID\Inc >> usbd_customhid.h file, change CUSTOM_HID_EPIN_SIZE and CUSTOM_HID_EPIN_SIZE >> from "0x02U" to >> "0x40".

usbd_customhid.h

```
47  #ifndef CUSTOM_HID_EPIN_SIZE
48  #define CUSTOM_HID_EPIN_SIZE                        0x40
49  #endif /* CUSTOM_HID_EPIN_SIZE */
50
51  #ifndef CUSTOM_HID_EPOUT_ADDR
52  #define CUSTOM_HID_EPOUT_ADDR                       0x01U
53  #endif /* CUSTOM_HID_EPOUT_ADDR */
54
55  #ifndef CUSTOM_HID_EPOUT_SIZE
56  #define CUSTOM_HID_EPOUT_SIZE                       0x40
57  #endif /* CUSTOM_HID_EPOUT_SIZE*/
```

- Also replace line 107 so that OutEvent expects a buffer instead of 2 bytes (uint8_t*)

usbd_customhid.h

```
107   //int8_t(*OutEvent)(uint8_t event_idx, uint8_t state);
108   int8_t (*OutEvent)(uint8_t*);
```

- You can also speed up the frequency of queries from the host. Replace it with the 2 images below:

usbd_customhid.c

```
184  CUSTOM_HID_FS_BINTERVAL,                    /* bInterval: Polling Interval */
185  /* 34 */
186
187  0x07,                                       /* bLength: Endpoint Descriptor si
188  USB_DESC_TYPE_ENDPOINT,                     /* bDescriptorType: */
189  CUSTOM_HID_EPOUT_ADDR,                      /* bEndpointAddress: Endpoint Addr
190  0x03,                                       /* bmAttributes: Interrupt endpoin
191  CUSTOM_HID_EPOUT_SIZE,                      /* wMaxPacketSize: 2 Bytes max  */
192  0x00,
193  CUSTOM_HID_FS_BINTERVAL,                    /* bInterval: Polling Interval */
```

```
183  0x00,
184  0x1,                          /* bInterval: Polling Interval */
185  /* 34 */
186
187  0x07,                                       /* bLength: End
188  USB_DESC_TYPE_ENDPOINT,                     /* bDescriptorT
189  CUSTOM_HID_EPOUT_ADDR,                      /* bEndpointAdd
190  0x03,                                       /* bmAttributes
191  CUSTOM_HID_EPOUT_SIZE,                      /* wMaxPacketSi
192  0x00,
193  0x1,                          /* bInterval: Polling Interval */
```

**It was a variable, so I changed it to 0x1, because the value is used elsewhere. Do the same! - Align the source code with the header file:

usbd_customhid.c

```
633   ((USBD_CUSTOM_HID_ItfTypeDef *)pdev->pUserData[pdev->classId])->OutEvent(hhid->Report_buf[0],
634   hhid->Report_buf[1]);    too many arguments in function call
```

```
685     ((USBD_CUSTOM_HID_ItfTypeDef *)pdev->pUserData[pdev->classId])->OutEvent(hhid->Report_buf[0],
686     hhid->Report_buf[1]);    too many arguments in function call
687     hhid->IsReportAvailable = 0U;
```

```
632   Naked till the end of the application processing */
633   ((USBD_CUSTOM_HID_ItfTypeDef *)pdev->pUserData[pdev->classId])->OutEvent(hhid->Report_buf);
```

- Insert a buffer on line ~65 of the usbd_custom_hid_if.c file.

usbd_custom_hid_if.c

```
C usbd_custom_hid_if.c U ×
USB_DEVICE > App > C usbd_custom_hid_if.c > buffer
64    /* USER CODE BEGIN PRIVATE_DEFINES */
65    uint8_t buffer[0x40];
66    /* USER CODE END PRIVATE_DEFINES */
```

- CUSTOM_HID_OutEvent_FS function parameters should also be modified!
**For example, I commented out what I didn't need, and included a memory copy function (!)

```
127   static int8_t CUSTOM_HID_OutEvent_FS(uint8_t *state);
128
```

```
static int8_t CUSTOM_HID_OutEvent_FS(uint8_t *state)
{
  /* USER CODE BEGIN 6 */
  // UNUSED(event_idx);
  memcpy(buffer, state, 0x40);
  UNUSED(state);
```

- The reportdesc must also be extended to be recognized by the PC as a USB HID device.
Below is a table with the code snippet:

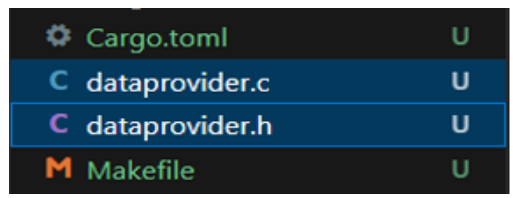
```
90   /** Usb HID report descriptor. */
91   __ALIGN_BEGIN static uint8_t CUSTOM_HID_ReportDesc_FS[USBD_CUSTOM_HID_REPORT_DESC_SIZE]
92       {
93           /* USER CODE BEGIN 0 */
94           0x00,
95           /* USER CODE END 0 */
96           0xC0 /*     END_COLLECTION              */
97   };
98
```

```
__ALIGN_BEGIN static uint8_t CUSTOM_HID_ReportDesc_FS
  {
    /* USER CODE BEGIN 0 */
    0x06, 0x00, 0xff, // Usage Page(Undefined)
    0x09, 0x01,       // USAGE(Undefined)
    0xa1, 0x01,       // COLLECTION (Application)
    0x15, 0x00,       // LOGICAL_MINIMUM(0)
    0x26, 0xff, 0x00, // LOGICAL_MAXIMUM(255)
    0x75, 0x08,       // REPORT_SIZE(8)
    0x95, 0x40,       // REPORT_COUNT(64)
    0x09, 0x01,       // USAGE (Undefined)
    0x81, 0x02,       // INPUT (Data,Var,Abs)
    0x95, 0x40,       // REPORT_COUNT(64)
    0x09, 0x01,       // USAGE (Undefined)
    0x91, 0x02,       // OUTPUT (Data,Var,Abs)
    0x95, 0x01,       // REPORT_COUNT(1)
    0x09, 0x01,       // USAGE (Undefined)
    0xb1, 0x02,       // FEATURE (Data,Var,Abs)
    /* USER CODE END 0 */
    0xC0 /*     END_COLLECTION              */
};
```

```
/* USER CODE BEGIN 0 */
    0x06, 0x00, 0xff, // Usage Page(Undefined)
    0x09, 0x01,       // USAGE(Undefined)
    0xa1, 0x01,       // COLLECTION (Application)
    0x15, 0x00,       // LOGICAL_MINIMUM(0)
    0x26, 0xff, 0x00, //
LOGICAL_MAXIMUM(255)
    0x75, 0x08,       // REPORT_SIZE(8)
    0x95, 0x40,       // REPORT_COUNT(64)
    0x09, 0x01,       // USAGE (Undefined)
    0x81, 0x02,       // INPUT (Data,Var,Abs)
    0x95, 0x40,       // REPORT_COUNT(64)
    0x09, 0x01,       // USAGE (Undefined)
    0x91, 0x02,       // OUTPUT (Data,Var,Abs)
    0x95, 0x01,       // REPORT_COUNT(1)
    0x09, 0x01,       // USAGE (Undefined)
    0xb1, 0x02,       // FEATURE (Data,Var,Abs)
    /* USER CODE END 0 */
    0xC0 /*     END_COLLECTION              */
```

I created 2 new files .h and .c singleton dataprovider.

Do the same!



Header: dataprovider.h

```
#ifndef DATAPROVIDER_H
#define DATAPROVIDER_H

struct dataprovider
{
  int speed;
};

// Accessing a global instance of the data structure
struct dataprovider *getDataprovider();

// Create and initialise a single instance
void initDataprovider();

int getSpeed();
void setSpeed(int value);

#endif
```

Source: dataprovider.c

```
#include "dataprovider.h"

static struct dataprovider instance;

struct dataprovider *getDataprovider()
{
   return &instance;
}

void initDataprovider()
{
   instance.speed = 250;
}

int getSpeed()
{
   struct dataprovider *dp = getDataprovider();
   return dp->speed;
}

void setSpeed(int value)
{
   struct dataprovider *dp = getDataprovider();
   dp->speed = value;
}
```

- Include it in main.c and create it at startup!

main.c

```c
/* Private includes ----------
/* USER CODE BEGIN Includes */
#include "../../dataprovider.h"
/* USER CODE END Includes */

```

```c
 */
int main(void)
{
  /* USER CODE BEGIN 1 */
  initDataprovider();
  /* USER CODE END 1 */
```

After that, here's a simple code: write a led blinky program:

main.c

```c
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  /* USER CODE END WHILE */
  HAL_Delay(getSpeed());
  HAL_GPIO_TogglePin(LED1_GPIO_Port, LED1_Pin);
  /* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
```

Write a simple RUST function that returns different speeds for ASCII 1-2-3 characters.

lib.rs

```rust
#![allow(non_snake_case)]
#![no_std]

extern crate cortex_m_rt;
extern crate panic_halt;

#[no_mangle]
pub extern "C" fn get_speed_from_rust(buffer: &mut [u8; 0x40]) -> u8 {
    match buffer[1] {
        49 => 64,  // ASCII 1
        50 => 128, // ASCII 2
        51 => 255, // ASCII 3
        _ => 250,
    }
}
```

Once you have it, download cbindgent with "cargo install --force cbindgen".

You can read documentation about it here: (https://github.com/mozilla/cbindgen)

To generate the C code, I called the following command:

"cbindgen --crate Balluff_USB_HID --output rust_wrapper.h --lang c"

```
> cbindgen --crate Balluff_USB_HID --output rust_wrapper.h --lang c
```

They must be included

usbd_custom_hid_if.c

```
USB_DEVICE
  App
    usb_device.c        U
    usb_device.h        U
    usbd_custom_hid_if.c U
    usbd_custom_hid_if.h U
    usbd_desc.c         U
```

```c
/* Includes ------------------------------------------------------------------
#include "usbd_custom_hid_if.h"

/* USER CODE BEGIN INCLUDE */
#include "../../dataprovider.h"
#include "../../rust_wrapper.h"
/* USER CODE END INCLUDE */
```

It is also necessary to create the dataprovider instance.

```c
/* Private variables ---------------------------------------------------------
struct dataprovider Dataprovider;
/* USER CODE END PV */
```

Let's complete things further down the line, based on the image cut: we get the speed back.

We set it up for the dataprovider. Then it returns an ASCII "ok".

```c
static int8_t CUSTOM_HID_OutEvent_FS(uint8_t *state)
{
  /* USER CODE BEGIN 6 */
  // UNUSED(event_idx);
  memcpy(buffer, state, 0x40);
  uint8_t num = get_speed_from_rust(&buffer);
  setSpeed(num);

  memset(buffer, 0, 0x40);
  buffer[1] = 79; // O
  buffer[2] = 75; // K
  USBD_CUSTOM_HID_SendReport(&hUsbDeviceFS, buffer, 0x40);
```

(!) Here is the makefile:

Makefile

```
# ChangeLog :
    2023-08-28 - Extended with RUST files by SzaboBenyo
#   2017-02-10 - Several enhancements + project update mode
#   2015-07-22 - first version
# ----------------------------------------------
```

1. changelog

```
Middlewares/ST/STM32_USB_Device_Library/Class/CustomHID/Src/usbd_customhid.c \
dataprovider.c
```

2. Extending C source files

```
target
  release
  thumbv7em-none-eabihf
    release
      .fingerprint
      build
      deps
      examples
      incremental
      .cargo-lock
      libBalluff_USB_HID.a
      libBalluff_USB_HID.d
```

```
# link script
LDSCRIPT = STM32F412ZGTx_FLASH.ld

# libraries
LIBS = -lc -lm -lnosys
LIBDIR =
LDFLAGS = $(MCU) -specs=nano.specs -T$(LDSCRIPT) $(LIBDIR) $(

RUST_PROJECT_PATH = ./target/thumbv7em-none-eabihf/release

RUST_PROJECT_NAME = libBalluff_USB_HID

# default action: build all
all: $(BUILD_DIR)/$(TARGET).elf $(BUILD_DIR)/$(TARGET).hex $(
```

3. Extending Flags with RUST

```
# default action: build all
all: $(BUILD_DIR)/$(TARGET).elf $(BUILD_DIR)/$(TARGET).hex $(BUILD_DIR)/$(TARGET).bin $(RUST_PROJECT_PATH)/$(RUST_PROJECT_NAME).a
```

4. When building, we also want to compile the RUST library

```
# BUILD RUST PROJECT FOR STM32F412ZG MCU LIKE THIS --> //cargo build --target thumbv7em-none-eabihf --release
$(RUST_PROJECT_PATH)/$(RUST_PROJECT_NAME):
    cargo build --target thumbv7em-none-eabihf --release
```

5. If there is none, this is how to do it

```
$(BUILD_DIR)/$(TARGET).elf: $(OBJECTS) Makefile
    $(CC) $(OBJECTS) $(LDFLAGS) $(RUST_PROJECT_PATH)/$(RUST_PROJECT_NAME).a -o $@
    $(SZ) $@
```

6. Extension with RUST

Testing... Successful at: 2023.09.20 13:31

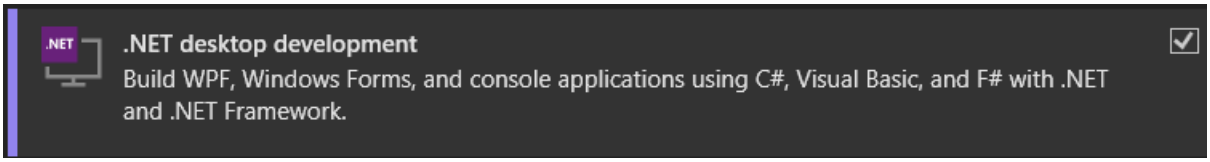
5. Create a C# app on PC in Visual Studio - WinFormApp .NET extensions + HIDSharp

Under the name Training materials you will find the files for the STM USB course:
https://www.st.com/content/st_com/en/support/learning/stm32-education/stm32-moocs/STM32-USB-training.html

It's the same, but you have direct access to Google Drive:
https://drive.google.com/file/d/1sjU9iNvh_khDZHDM9Qau03PGKwMd4u7U/view

Or you can download it from me, it's in the Debug folder:
https://github.com/szabobenyo/USB_HID_C_and_RUST

But if you want to make your own WindowsFormApp, install this:



The program uses the https://github.com/IntergatedCircuits/HidSharp directory. It is simple.

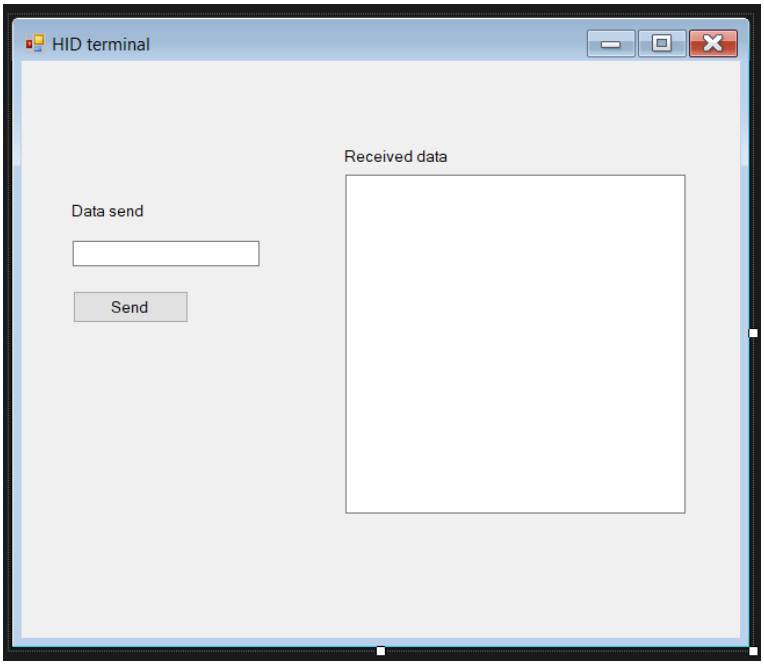
This is what the code looks like:

```
using System;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using HidSharp;


namespace HID_terminal
{
    3 references
    public partial class Form1 : Form
    {
        HidDeviceLoader loader;
        HidDevice device;
        HidStream stream;
        byte []bytes;
        1 reference
        public Form1()
        {
            InitializeComponent();
            loader = new HidDeviceLoader();
            device = loader.GetDevices(1155, 22352).FirstOrDefault();
            if (device == null)
            {
                Console.WriteLine("Failed to open device.");
                MessageBox.Show("Failed to open device.", "Error",
                MessageBoxButtons.OK, MessageBoxIcon.Error);
                Environment.Exit(1);
            }

        }
```

```
private void button_send_Click(object sender, EventArgs e)
{
    if (!device.TryOpen(out stream))
    {
        Console.WriteLine("Failed to open device."); MessageBox.Show("Failed to open device.", "Error",
        MessageBoxButtons.OK, MessageBoxIcon.Error); Environment.Exit(1);
    }

    using (stream)
    {
        int n = 0;

            bytes = new byte[device.MaxInputReportLength];
            int count=0;
            var message = new byte[64];
            ASCIIEncoding.ASCII.GetBytes(textBoxSend.Text, 0, textBoxSend.Text.Length, message, 2);
            message[0] = 0;
            message[1] = (byte)textBoxSend.Text.Length;
            stream.Write(message, 0, 2+textBoxSend.Text.Length);
            try
            {
                count = stream.Read(bytes);
            }
            catch (TimeoutException)
            {
                Console.WriteLine("Read timed out.");
            }

            if (count > 0)
            {
                this.BeginInvoke(new EventHandler(DoUpdate));
            }

}
}
```

```
1 reference
private void DoUpdate(object sender, System.EventArgs e)
{
    string s = Encoding.UTF8.GetString(bytes, 2, 2+bytes[1]);
    textBoxRecieved.AppendText(s);
    textBoxRecieved.AppendText("\n");

}
```

The form app itself:



Resources:

Visual Studio Code: https://code.visualstudio.com/download

RUST installer: https://www.rust-lang.org/tools/install

Rust YT video 2: https://www.youtube.com/watch?v=BU1LYFkpJuk

Rust YT video 1: https://www.youtube.com/watch?v=yo4kWLtSPCY

cubeMX: https://www.st.com/en/development-tools/stm32cubemx.html

Makefile toolchain: https://www.youtube.com/watch?v=ZP2fd2qatj0

MSYS2: https://www.msys2.org/

List of targets: https://docs.rust-embedded.org/book/intro/install.html

RUST install: https://docs.rust-embedded.org/book/intro/install.html

Embedded book: https://docs.rust-embedded.org/book/intro/install/windows.html

GNU Toolchain: https://developer.arm.com/downloads/-/gnu-rm

OpenOCD: https://github.com/xpack-dev-tools/openocd-xpack/releases/

OpenOCD needs for MCUs: https://www.st.com/en/development-tools/stsw-link009.html

Custom USB HID course video: https://www.youtube.com/watch?v=3JGRt3BFYrM

Cbindgen: https://github.com/mozilla/cbindgen

Training materials: https://www.st.com/content/st_com/en/support/learning/stm32-education/stm32-moocs/STM32-USB-training.html

Training materials google drive:
https://drive.google.com/file/d/1sjU9iNvh_khDZHDM9Qau03PGKwMd4u7U/view