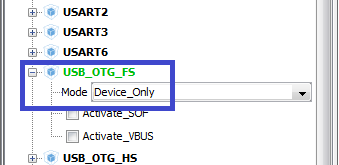
**HAL. USB. Custom HID**

We continue to work with the USB bus and also continue to work with the USB HID class.

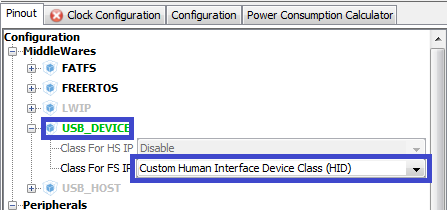
Only today we will try to work with a non-standard class and write some kind of HID class.

The project is created from the project TEST002. Call it USB\_DEVICE\_CUSTOM\_HID. Run the project in the Cube, do not turn it off. and timers

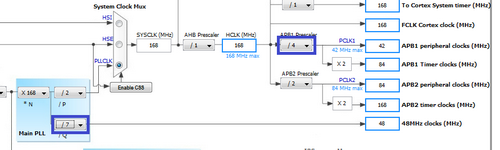
Turn on USB\_OTG\_FS in Device\_Only mode.



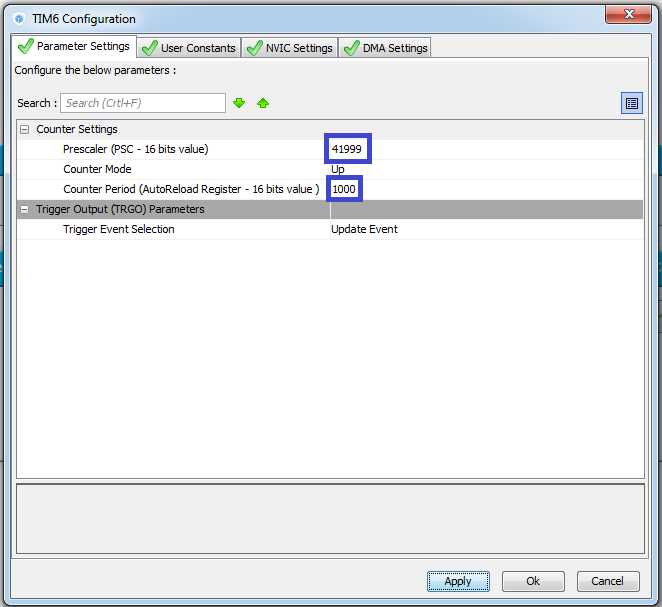
In the USB Device, select the Custom Human Interface Device Class (HID)



In Clock Configuration, select the following dividers (click on the image to increase the size)

[](http://narodstream.ru/wp-content/uploads/2016/12/image01_0862.png)

In Configuration the timer will be made a little slower.



Generate and run the project.

We will collect the project and set up the programmer for auto-cutting.

Let's run through the controller and see that the device is installed with an error (code 10).

Before writing descriptors, let's repeat what kinds they are

Let's see here[http://www.keil.com/pack/doc/mw/usb/html/index.html](https://www.google.com/url?q=http://www.keil.com/pack/doc/mw/usb/html/index.html&sa=D&ust=1480683574778000&usg=AFQjCNEWKE7rMmmAFHlDYO0cIL3L5KEYlQ)

Let's open on the left page **USB Concepts-> USB Descriptors**

Let's look at descriptors in order.

Also we download the convenient utility for forming the descriptor of the HID Descriptor Tool report on the site <http://www.usb.org/developers/hidpage/>

Run it and see its abilities

In the file usbd\_customhid.c we look at interface, configuration, etc. descriptors.

In the file usbd\_custom\_hid\_if.c we see an empty report descriptor.

 \_\_ALIGN\_BEGIN static uint8\_t CUSTOM\_HID\_ReportDesc\_FS [USBD\_CUSTOM\_HID\_REPORT\_DESC\_SIZE] \_\_ALIGN\_END =

{

  / \* USER CODE BEGIN 0 \* /

  0x00,

  / \* USER CODE END 0 \* /

  0xC0 / \* END\_COLLECTION \* /

Let's write the descriptor of the report from our device.

  / \* USER CODE BEGIN 0 \* /

**0x06, 0x00, 0xff, // USAGE\_PAGE (Generic Desktop)**

**0x09, 0x01, // USAGE (Vendor Usage 1)**

**0xa1, 0x01, // COLLECTION (Application)**

**0x85, 0x01, // REPORT\_ID (1)**

**0x09, 0x01, // USAGE (Vendor Usage 1)**

**0x15, 0x00, // LOGICAL\_MINIMUM (0)**

**0x25, 0x01, // LOGICAL\_MAXIMUM (1)**

**0x75, 0x08, // REPORT\_SIZE (8)**

**0x95, 0x01, // REPORT\_COUNT (1)**

**0xB1, 0x82, // FEATURE (Data, Var, Abs, Vol)**

**0x85, 0x01, // REPORT\_ID (1)**

**0x09, 0x01, // USAGE (Vendor Usage 1)**

**0x91, 0x82, // OUTPUT (Data, Var, Abs, Vol)**

**0x85, 0x02, // REPORT\_ID (2)**

**0x09, 0x02, // USAGE (Vendor Usage 2)**

**0x15, 0x00, // LOGICAL\_MINIMUM (0)**

**0x25, 0x01, // LOGICAL\_MAXIMUM (1)**

**0x75, 0x08, // REPORT\_SIZE (8)**

**0x95, 0x01, // REPORT\_COUNT (1)**

**0xB1, 0x82, // FEATURE (Data, Var, Abs, Vol)**

**0x85, 0x02, // REPORT\_ID (2)**

**0x09, 0x02, // USAGE (Vendor Usage 2)**

**0x91, 0x82, // OUTPUT (Data, Var, Abs, Vol)**

**0x85, 0x03, // REPORT\_ID (3)**

**0x09, 0x03, // USAGE (Vendor Usage 3)**

**0x15, 0x00, // LOGICAL\_MINIMUM (0)**

**0x25, 0x01, // LOGICAL\_MAXIMUM (255)**

**0x75, 0x08, // REPORT\_SIZE (8)**

**0x95, 0x01, // REPORT\_COUNT (1)**

**0xB1, 0x82, // FEATURE (Data, Var, Abs, Vol)**

**0x85, 0x03, // REPORT\_ID (3)**

**0x09, 0x03, // USAGE (Vendor Usage 3)**

**0x91, 0x82, // OUTPUT (Data, Var, Abs, Vol)**

**0x85, 0x04, // REPORT\_ID (4)**

**0x09, 0x04, // USAGE (Vendor Usage 4)**

**0x15, 0x00, // LOGICAL\_MINIMUM (0)**

**0x25, 0x01, // LOGICAL\_MAXIMUM (255)**

**0x75, 0x08, // REPORT\_SIZE (8)**

**0x95, 0x01, // REPORT\_COUNT (1)**

**0xB1, 0x82, // FEATURE (Data, Var, Abs, Vol)**

**0x85, 0x04, // REPORT\_ID (4)**

**0x09, 0x04, // USAGE (Vendor Usage 4)**

**0x91, 0x82, // OUTPUT (Data, Var, Abs, Vol)**

**0x85, 0x05, // REPORT\_ID (5)**

**0x09, 0x05, // USAGE (Vendor Usage 5)**

**0x75, 0x08, // REPORT\_SIZE (8)**

**0x95, 0x04, // REPORT\_COUNT (4)**

**0x81, 0x82, // INPUT (Data, Var, Abs, Vol)**

**0x85, 0x06, // REPORT\_ID (6)**

**0x09, 0x06, // USAGE (Vendor Usage 6)**

**0x75, 0x08, // REPORT\_SIZE (8)**

**0x95, 0x04, // REPORT\_COUNT (4)**

**0x81, 0x82, // INPUT (Data, Var, Abs, Vol)**

  / \* USER CODE END 0 \* /

  0xC0 / \* END\_COLLECTION \* /

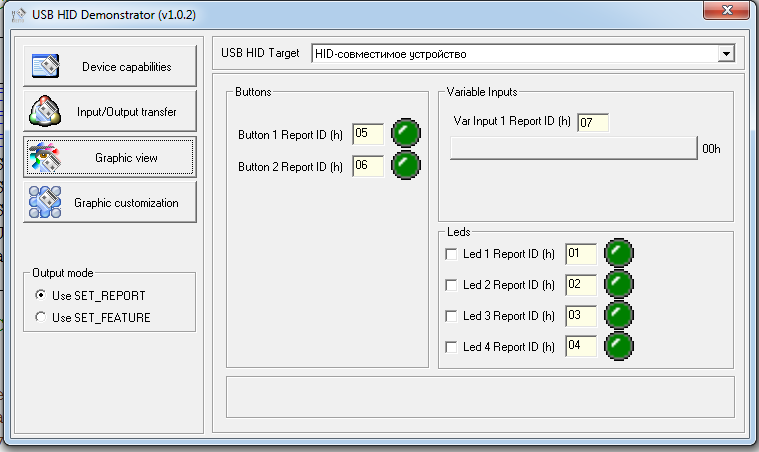
After that, we see a compilation error, because we need to change the size of our descriptor. Let's change it in the Cube. We close the project and in Configuration USB DEVICE we introduce 108.

Again, generate the code, open the project, collect it and we'll sew the controller.

As we can see, the error (code 10) in the Device Manager has disappeared.

Also download the USB HID Demonstrator utility from st.com for convenient viewing of messages from the device by typing this line (USB HID Demonstrator) in the search line, install it and run it. We will see that our homemade device was found and maintained there. This is already good.

We will configure the utility as follows



Add a file connection with functions in main.c

/ \* USER CODE BEGIN Includes \* /

#include "main.h"

**#include "usbd\_customhid.h"**

We also add variables

/ \* USER CODE BEGIN PV \* /

/ \* Private variables ------------------- \* /

**extern USBD\_HandleTypeDef hUsbDeviceFS;**

**uint8\_t dataToSend [5];**

**uint8\_t btn\_stat = 0;**

Let's start the timer

  / \* USER CODE BEGIN 2 \* /

**HAL\_TIM\_Base\_Start\_IT (& htim6);**

  / \* USER CODE END 2 \* /

Add the function-processor interrupt from the timer.

/ \* USER CODE BEGIN 4 \* /

**void HAL\_TIM\_PeriodElapsedCallback (TIM\_HandleTypeDef \* htim6)**

**{**

**if (btn\_stat == 0) {dataToSend [0] = 6; dataToSend [1] = 1;}**

**else if (btn\_stat == 1) {dataToSend [0] = 6; dataToSend [1] = 0;}**

**else if (btn\_stat == 2) {dataToSend [0] = 5; dataToSend [1] = 1;}**

**else if (btn\_stat == 3) {dataToSend [0] = 5; dataToSend [1] = 0;}**

**dataToSend [2] = 'H';**

**dataToSend [3] = 'I';**

**dataToSend [4] = 'D';**

**USBD\_CUSTOM\_HID\_SendReport (& hUsbDeviceFS, dataToSend, 5);**

**btn\_stat ++;**

**if (btn\_stat> 3) btn\_stat = 0;**

**}**

Remove everything from an infinite loop

  / \* USER CODE BEGIN 3 \* /

~~if (HAL\_GPIO\_ReadPin (GPIOA, GPIO\_PIN\_0) == GPIO\_PIN\_SET)~~

~~{~~

~~HAL\_TIM\_Base\_Start (& htim6);~~

~~HAL\_TIM\_Base\_Start\_IT (& htim6);~~

~~}~~

~~else~~

~~{~~

~~tim6\_counter = 0;~~

~~HAL\_TIM\_Base\_Stop (& htim6);~~

~~HAL\_TIM\_Base\_Stop\_IT (& htim6);~~

~~HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_12 | GPIO\_PIN\_13 | GPIO\_PIN\_14 | GPIO\_PIN\_15, GPIO\_PIN\_RESET);~~

~~}~~

  }

  / \* USER CODE END 3 \* /

Also remove all of the file

void TIM6\_DAC\_IRQHandler (void)

{

  / \* USER CODE BEGIN TIM6\_DAC\_IRQn 0 \* /

  / \* USER CODE END TIM6\_DAC\_IRQn 0 \* /

  HAL\_TIM\_IRQHandler (& htim6);

  / \* USER CODE BEGIN TIM6\_DAC\_IRQn 1 \* /

~~switch (tim6\_counter)~~

~~{~~

~~case 0:~~

~~HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_12 | GPIO\_PIN\_13 | GPIO\_PIN\_14 | GPIO\_PIN\_15, GPIO\_PIN\_RESET);~~

~~HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_12, GPIO\_PIN\_SET);~~

~~break;~~

~~case 1:~~

~~HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_12, GPIO\_PIN\_RESET);~~

~~HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_13, GPIO\_PIN\_SET);~~

~~break;~~

~~case 2:~~

~~HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_13, GPIO\_PIN\_RESET);~~

~~HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_14, GPIO\_PIN\_SET);~~

~~break;~~

~~case 3:~~

~~HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_14, GPIO\_PIN\_RESET);~~

~~HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_15, GPIO\_PIN\_SET);~~

~~break;~~

~~}~~

~~if (tim6\_counter <3) tim6\_counter ++;~~

~~else tim6\_counter = 0;~~

  / \* USER CODE END TIM6\_DAC\_IRQn 1 \* /

}

Also remove from there this variable

/ \* USER CODE BEGIN 0 \* /

~~extern uint8\_t tim6\_counter;~~

/ \* USER CODE END 0 \* /

Let's collect the code. Let's run the controller and check with the utility.

In the file usbd\_custom\_hid\_if.c we add a buffer

/ \* USER CODE BEGIN PRIVATE\_VARIABLES \* /

**uint8\_t dataToReceive [2];**

/ \* USER CODE END PRIVATE\_VARIABLES \* /

Also we will make changes in the function

static int8\_t CUSTOM\_HID\_OutEvent\_FS (uint8\_t event\_idx, uint8\_t state)

{

  / \* USER CODE BEGIN 6 \* /

**USBD\_CUSTOM\_HID\_HandleTypeDef \* hhid = (USBD\_CUSTOM\_HID\_HandleTypeDef \*) hUsbDeviceFS.pClassData;**

**for (uint8\_t i = 0; i <4; i ++)**

**{**

**dataToReceive [i] = hhid-> Report\_buf [i];**

**}**

**if ((dataToReceive [0] == 1) && (dataToReceive [1] == 1))**

**{**

**HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_12, GPIO\_PIN\_SET);**

**}**

**if ((dataToReceive [0] == 1) && (dataToReceive [1] == 0))**

**{**

**HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_12, GPIO\_PIN\_RESET);**

**}**

**if ((dataToReceive [0] == 2) && (dataToReceive [1] == 1))**

**{**

**HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_13, GPIO\_PIN\_SET);**

**}**

**if ((dataToReceive [0] == 2) && (dataToReceive [1] == 0))**

**{**

**HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_13, GPIO\_PIN\_RESET);**

**}**

**if ((dataToReceive [0] == 3) && (dataToReceive [1] == 1))**

**{**

**HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_14, GPIO\_PIN\_SET);**

**}**

**if ((dataToReceive [0] == 3) && (dataToReceive [1] == 0))**

**{**

**HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_14, GPIO\_PIN\_RESET);**

**}**

**if ((dataToReceive [0] == 4) && (dataToReceive [1] == 1))**

**{**

**HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_15, GPIO\_PIN\_SET);**

**}**

**if ((dataToReceive [0] == 4) && (dataToReceive [1] == 0))**

**{**

**HAL\_GPIO\_WritePin (GPIOD, GPIO\_PIN\_15, GPIO\_PIN\_RESET);**

**}**

  return (0);

  / \* USER CODE END 6 \* /

}

Let's check in practice using the utility.