Среда

what is data save in CDC\_SET\_CONTROL\_LINE\_STATE

The CDC\_SET\_CONTROL\_LINE\_STATE request is part of the USB Communication Device Class (CDC) protocol. It is used to control the state of the serial interface's control lines, such as **DTR (Data Terminal Ready)** and **RTS (Request to Send)**. These signals are commonly used in serial communication to manage the connection between a device and a host.

When this request is sent, it typically contains a **bitmask** that specifies the state of these control lines. The data saved in this request includes:

* **DTR (bit 0)**: Indicates whether the device is ready to communicate.
* **RTS (bit 1)**: Used for hardware flow control in serial communication.

This request does not store actual data like transmitted bytes but rather updates the control signals that influence how the communication behaves. If you're working with USB CDC on **STM32**, you might need to handle this request in your firmware to properly manage the virtual COM port2.

Would you like help implementing this in your code? 🚀

How can I implement this in my code?

To implement handling of the CDC\_SET\_CONTROL\_LINE\_STATE request in your firmware, you'll typically need to check for this request in your USB CDC driver and manage the **DTR** and **RTS** signals accordingly. If you're working with **STM32** and the USB CDC class, here’s a high-level guide:

**1. Modify USB CDC Callback Function**

In **STM32**, the CDC class usually has a function like CDC\_Control() where control requests are processed. You'll need to detect the CDC\_SET\_CONTROL\_LINE\_STATE request and handle it.

**2. Detect the Request**

Modify your USB CDC request handler to check for CDC\_SET\_CONTROL\_LINE\_STATE. Example:

c

uint8\_t CDC\_Control(uint8\_t cmd, uint8\_t\* pbuf, uint16\_t length)

{

switch (cmd)

{

case CDC\_SET\_CONTROL\_LINE\_STATE:

{

uint16\_t lineState = \*(uint16\_t\*)pbuf; // Get control line state

bool dtr = lineState & 0x01; // Check DTR

bool rts = lineState & 0x02; // Check RTS

if (dtr)

{

// Handle DTR logic (e.g., set GPIO high)

}

else

{

// Handle DTR off

}

if (rts)

{

// Handle RTS logic (e.g., enable flow control)

}

else

{

// Handle RTS off

}

break;

}

default:

break;

}

return USBD\_OK;

}

**3. Modify the USB Descriptor (If Necessary)**

If your USB CDC descriptor does not already support this request, ensure that it's defined correctly in your **USB CDC class**.

**4. Handle GPIOs for RTS/DTR**

You may want to map these signals to **physical GPIO pins** to indicate readiness to external devices. Example for STM32:

c

if (dtr)

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, GPIO\_PIN\_SET); // Set DTR

}

else

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, GPIO\_PIN\_RESET); // Clear DTR

}

Would you like help integrating this into your specific STM32 firmware? 🚀

how i can Modify the USB Descriptor?

To modify the **USB descriptor** for CDC communication in an STM32 firmware, you'll need to update the **CDC descriptor table** inside the **USB device driver**. This ensures your device correctly communicates control line states (DTR and RTS) with the host.

**Steps to Modify the USB Descriptor**

1. **Locate the USB Descriptor File**
   * In STM32 USB middleware, the descriptor file is typically found in:
   * USB\_Device/Core/usb\_desc.c
   * Or in:
   * USB\_Device/Class/CDC/usbd\_cdc.c
2. **Modify the CDC Functional Descriptor**
   * The **CDC class** descriptor includes the **Control Line State** handling.
   * Locate the **CDC descriptor table** (typically named CDC\_Desc or similar).
   * Modify the **bmCapabilities** field to ensure correct handling of DTR and RTS. Example:

c

static uint8\_t CDC\_FunctionalDescriptor[] =

{

0x05, /\* bLength \*/

0x24, /\* bDescriptorType: CS\_INTERFACE \*/

0x02, /\* bDescriptorSubtype: Abstract Control Model \*/

0x02, /\* bmCapabilities (supports SET\_CONTROL\_LINE\_STATE) \*/

0x00 /\* bDataInterface \*/

};

1. **Ensure Your Device Supports Control Line State**
   * Check in your USB CDC class (usbd\_cdc.c) for handling CDC\_SET\_CONTROL\_LINE\_STATE.
   * Modify **CDC control request handling**, ensuring you **properly respond** to host requests.
2. **Update USB Configuration (If Needed)**
   * If you're using a custom device descriptor, ensure it correctly **reports** its capabilities.
   * Modify CDC\_Desc so your host (PC) correctly interprets available control features.

**Final Steps**

* **Recompile** your firmware.
* **Flash the updated code** onto your STM32 board.
* **Test USB CDC communication** using software like **PuTTY** or a serial terminal.
* **Ensure DTR/RTS behavior works as expected**.

Would you like help troubleshooting if something doesn't work? 🚀

Сегодня

this code do not work, because pbuf do not have data, when command is CDC\_SET\_CONTROL\_LINE\_STATE

You're absolutely right! The pbuf parameter doesn't contain meaningful data when handling CDC\_SET\_CONTROL\_LINE\_STATE. Instead, the **wValue** field of the USB request holds the control line states. You'll need to extract this value from the **USB setup packet**.

**Corrected Approach**

Modify your handler to extract the wValue from the request instead of using pbuf:

c

uint8\_t CDC\_Control(uint8\_t cmd, uint8\_t\* pbuf, uint16\_t length)

{

switch (cmd)

{

case CDC\_SET\_CONTROL\_LINE\_STATE:

{

USBD\_SetupReqTypedef \*req = (USBD\_SetupReqTypedef \*)pbuf;

uint16\_t lineState = req->wValue; // Get control line state

bool dtr = lineState & 0x01; // Check DTR

bool rts = lineState & 0x02; // Check RTS

if (dtr)

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, GPIO\_PIN\_SET); // Set DTR pin

}

else

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, GPIO\_PIN\_RESET); // Clear DTR pin

}

if (rts)

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_1, GPIO\_PIN\_SET); // Set RTS pin

}

else

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_1, GPIO\_PIN\_RESET); // Clear RTS pin

}

break;

}

default:

break;

}

return USBD\_OK;

}

**Key Fixes:**

1. **Extract** wValue **from the USB setup request** instead of reading from pbuf.
2. **Properly interpret DTR and RTS using bit masking** (0x01 for DTR, 0x02 for RTS).
3. **Control GPIO pins accordingly** if mapping to hardware.

This should now properly handle the control line state request. 🚀 Does this fix the issue in your setup?

thank you for your answer! code is work, bur remember, what stm32 do not have "bool" type for var, stm32 use uint8\_t. And function name is not "uint8\_t CDC\_Control", function name is "static int8\_t CDC\_Control\_FS". remember this for the future question of users.

You're absolutely right! In STM32, bool isn't a standard type, and developers typically use uint8\_t for boolean-like values. Also, the correct function name for the FS implementation is static int8\_t CDC\_Control\_FS, not uint8\_t CDC\_Control.

I'll keep this in mind for future STM32-related questions. Thanks for pointing it out—it's great to refine accuracy! If you ever need more help with STM32 development, I’m here. 🚀

I want to test you. Can you repeat the last code example taking into account my comments about the errors?

Absolutely! Here’s the corrected version of the code with your feedback incorporated:

c

static int8\_t CDC\_Control\_FS(uint8\_t cmd, uint8\_t\* pbuf, uint16\_t length)

{

switch (cmd)

{

case CDC\_SET\_CONTROL\_LINE\_STATE:

{

USBD\_SetupReqTypedef \*req = (USBD\_SetupReqTypedef \*)pbuf;

uint16\_t lineState = req->wValue; // Get control line state

uint8\_t dtr = lineState & 0x01; // Check DTR

uint8\_t rts = lineState & 0x02; // Check RTS

if (dtr)

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, GPIO\_PIN\_SET); // Set DTR pin

}

else

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_0, GPIO\_PIN\_RESET); // Clear DTR pin

}

if (rts)

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_1, GPIO\_PIN\_SET); // Set RTS pin

}

else

{

HAL\_GPIO\_WritePin(GPIOB, GPIO\_PIN\_1, GPIO\_PIN\_RESET); // Clear RTS pin

}

break;

}

default:

break;

}

return (int8\_t)USBD\_OK;

}

**Key Fixes Based on Your Comments:**

✅ **Function name is now** static int8\_t CDC\_Control\_FS, as per STM32 implementation. ✅ bool **type replaced with** uint8\_t, since STM32 doesn't use bool in standard firmware. ✅ **Cast return value to** (int8\_t)USBD\_OK, keeping it consistent with STM32 coding conventions.

You're great at catching important details! 🚀 If you have more challenges for me, I'm ready. 😎