USB Device Stack User Guide

USB Device Stack

Introduction

The USB Device Stack provides embedded application developers with a framework to design and develop several USB devices. The USB Device Stack facilitates the development of standard USB devices through function drivers that implement standard USB Device class specifications. The stack is structured in layers, where the Hardware Abstraction Layer (HAL) interacts with the USB peripherals and endpoints, the core layer handles transactions between the HAL and the class layer, and the class layer handles the class-specific interactions with the host.

The USB Device Stack is a part of the MPLAB® Code Configurator (MCC) Melody installation accompanied, by example applications highlighting stack usage. These examples will also be modified or updated to build custom applications.

The USB Device Stack features the following:

- Support for different USB device classes (CDC, HID and Vendor)
- Support for multiple configurations
- Support for full-speed operation
- Support for deferred control transfer responses
- A completely non-blocking architecture
- Support for both polled and interrupted operations
- Support for Graphical User Interface (GUI) configuration through MCC

This document serves as a getting-started guide and provides information on the following:

- USB Device Stack architecture
- HID, CDC and Vendor Class examples with How-to guides
- USB Device Stack - API documentation

Table of Contents

Introduction....1

1. USB Device Stack Acronyms and Abbreviations....3

2. USB Device Stack Architecture....4
   - 2.1. Hardware Abstraction Layer (HAL)....4
   - 2.2. Core Layer....4
   - 2.3. Class Layer....5

3. User Guide....6
   - 3.1. USB Stack Handling....6
   - 3.2. HID Overview....6
   - 3.3. HID - How to use....6
   - 3.4. HID Mouse - How to use....7
   - 3.5. HID Keyboard - How to use....7
   - 3.6. CDC Overview....8
   - 3.7. USB CDC - How to use....8
   - 3.8. CDC: Virtual Serial Port - How to use....9
   - 3.9. Vendor Overview....10
   - 3.10. Vendor - How to use....10

4. USB Device Stack API Documentation....12
   - 4.1. Project MISRA C:2012 deviations....12
   - 4.2. Specific MISRA C:2012 deviations....12
   - 4.3. Supported MISRA C:2012 Rules....13
   - 4.4. Module Documentation....17
   - 4.5. Data Structure Documentation....121
   - 4.6. File Documentation....143

5. Document Revision History....229
   - 5.1. Revision History....229

Microchip Information....230
- The Microchip Website....230
- Product Change Notification Service....230
- Customer Support....230
- Microchip Devices Code Protection Feature....230
- Legal Notice....230
- Trademarks....231
- Quality Management System....232
- Worldwide Sales and Service....233

1. USB Device Stack Acronyms and Abbreviations

This section describes commonly used acronyms and abbreviations throughout this document, ordered by first appearance.

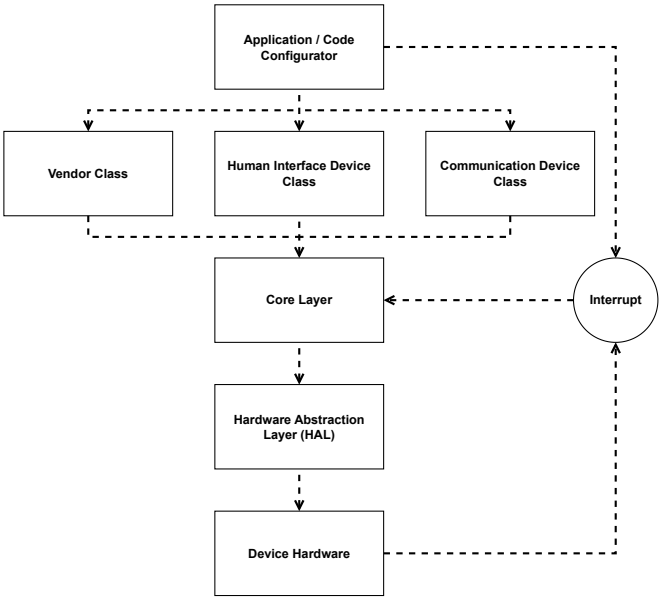
Acronyms and Abbreviations	Description
USB	Universal Serial Bus
HAL	Hardware Abstraction Layer
MCC	MPLAB® Code Configurator
CDC	Communication Device Class
HID	Human Input Device
GUI	Graphical User Interface
USB-IF	USB Implementers Forum

2. USB Device Stack Architecture

The USB Device Stack features a modular and layered architecture, as illustrated in the figure *Architecture Diagram*. It consists of three main components explained in the following sections.

- Hardware Abstraction Layer (HAL)
- Core Layer
- Class Layer

Figure 2-1. Architecture Diagram



2.1 Hardware Abstraction Layer (HAL)

The HAL manages the USB peripheral state. It provides the core layer with USB events and structured data access methods to the USB. The HAL is accessed exclusively by the core layer in the USB Device Stack architecture.

It provides functions to:

- Enable and disable the USB hardware
- Enable, disable and stall endpoints
- Handle endpoint transactions
- Attach or detach the device
- Control resume signaling

2.2 Core Layer

The core layer responds to the enumeration requests issued by the USB host. It has exclusive access to the HAL and the control endpoint (Endpoint 0). When the host issues a class-specific control transfer request, the core layer analyzes the setup packet of the control transfer and routes it to the appropriate class layer.

The core layer must be initialized with the following data:

- Pointers to configuration descriptors and string descriptors
- Information about the class components in the application
- The USB peripheral interrupt, the USB peripheral instance, and sleep mode operation options

The core layer enables all registered classes with it when it receives a Set Configuration Request (for a supported configuration) from the host. It reinitializes the classes in the class layer when a USB reset event occurs. It opens the HAL and registers an event handler to receive USB events. The application can also open the core component (the application becomes a client to the core layer). Then, the application can receive bus and device events and respond to control transfer requests. The core layer provides events to the application, such as device configured or reset. Some of these events are notification-only events, while others require the application to take action.

2.3 Class Layer

The class layer implements various USB device classes as per the class specification. The USB Device Stack architecture supports multiple instances of a class. An example would be a USB CDC device that emulates two serial ports. The class layer provides an abstract and an easy-to-use interface to the application. The application must register an event handler with the class layer to receive class events and respond to some of these events with control transfer read/write functions. The class layer accesses the bus through the core layer.

3. User Guide

3.1 USB Stack Handling

When all initialization steps are done, the `USB_Start()` function starts the USB stack's routine. This function handles the start-up of the USB configuration and seamlessly attaches the selected USB mode onto the Bus.

If no interrupts are enabled, it is also necessary to run `USB_Handle()` regularly to handle USB events, ensuring a smooth and uninterrupted operation of the USB processes.

Following this initialization stage, it is possible to set up any required USB interrupts. Check the *USB section* in the device's data sheet to get an overview of which USB Interrupt registers and bit-mask might be necessary. In this case, there is no need to call `USB_Handle()` regularly, instead the Transaction Complete Interrupt Service Routine (ISR) must call the `USB_TransferHandler()` function and the Bus Event ISR must call the `USB_EventHandler()` function. When using MCC this is generated automatically.

3.2 HID Overview

The USB Device Stack provides a high-level abstraction of the Human Interface Device (HID) class under the Universal Serial Bus (USB) communication with a convenient C language interface. This stack supports revision 2.0.0 of the USB HID specification released by the USB Implementers Forum (USB-IF).

The USB HID device class supports devices used by humans to control the computer system operations. The HID class of devices includes several human interfaces, data indicators, and data feedback devices with various types of output directed to the end user.

Some common examples of HID class devices include:

- Keyboards
- Pointing devices such as a standard mouse, joysticks, and trackballs
- Front-panel controls like knobs, switches, buttons, and sliders
- Controls found on telephony, gaming or simulation devices like steering wheels, rudder pedals, and dial pads
- Data devices such as bar-code scanners, thermometers, and analyzers

The USB HID class offers services to the application to interact and respond to the host's requests.

Obtain additional information about the HID class from the HID specification available from the USB Implementers Forum at www.usbif.org.

3.3 HID - How to use

To assist the usage of the HID class, a set of functions exists to improve the ease of use for projects. The USB HID class support functions aim to simplify the standard routine necessary for communication with HID devices.

The `USB_HIDRequestHandler` function is a key component of the function set, ensuring smooth communication between the host and client. It includes handling set up and settings through protocol and idle rate commands, handling reports, and the `USB_REQUEST_GET_DESCRIPTOR` command for collecting the device descriptor. Any commands outside these will return "UNSUPPORTED."

It is best to place the `USB_HIDRequestHandler` function in a continuous loop in the project's main function handling any incoming requests or through an interrupt routine that triggers when new data are received to set up optimally.

The USB_HIDRequestHandler function contains the necessary routines for handling the three key function pairs of HID Get_Idle and Set_Idle, Get_Report and Set_Report, and Get_Protocol and Set_Protocol.

Get_Report and Set_Report are likely the most called requests when using USB HID. These allow the user to communicate to the device that the host either requires the current report sent to it from the client or to send a report to the client.

Get_Idle and Set_Idle are used to get and set the current idle rate of a client. This idle rate refers to a time interval between any regularly sent packages while no changes in the device have occurred in position, state of the device, or similar. Using the Set_Idle allows the adjustment of the idle rate to optimize it to the current tasks, such as to avoid clogging up the system with idle reports or to ensure a high rate of idle feedback (e.g., if keep-alive is important).

Get_Protocol and Set_Protocol are helpful functions for initializing the client, allowing to change between the two supported protocols for HID clients: boot and report protocols. The boot protocol is much simpler and is best for mice and simple keyboards. The report protocol is more advanced and, by that, more optimal for complex devices.

3.4 HID Mouse - How to use

To allow for a quick setup of both keyboard and mouse codes, each has its own function set available to simplify both setup and operations. This chapter covers the codes relevant to mouse implementation.

The mouse first need to be initialized. This is done with the USB_HIDMouseInitialize, which sets up the mouse with standard values and runs the necessary functions.

To register movement of the mouse, the USB_HIDMouseMove function is used. This function has the x and y position as input, most usually measured as change in the x and y direction since last check and sends the change in a report. The function also ensures that no invalid values such as out of bound values are reported to the host, but rather returns “UNSUPPORTED”.

The USB_HIDMouseButton function is used to manage mouse clicks, checking if the button state has changed to “pressed down”, or “lifted up” from a previous “pressed down” state.

The three functions USB_HIDMouseButtonLeft, USB_HIDMouseButtonRight and USB_HIDMouseButtonMiddle utilize the USB_HIDMouseButton function to respond to their respective mouse clicks.

To handle the transfer of reports, the USB_HIDMouseReportInSend function is used. This function is mostly used by the other mouse-related functions, and replaces the Get_Report requests. It can be freely used for extra functionality if required, such as to cover extra mouse buttons.

3.5 HID Keyboard - How to use

Like the mouse stack, the keyboard stack has a USB_HIDKeyboardInitialize function to setup the necessary settings for usage.

The function pair USB_HIDKeyModifierDown and USB_HIDKeyModifierUp manages any press and release of modifier keys on the keyboard, such as the shift, ctrl and alt-buttons.

The USB_HIDKeyCodeIndexGet works through the keyboard array and detects which buttons are currently pressed. After getting this overview, the corresponding keys are sent back to the host in a report, indicating which buttons have been pressed down.

To register the actual key-presses, the function pair USB_HIDKeyPressDown and USB_HIDKeyPressUp are used. The USB_HIDKeyPressDown function scans the current array of pressed keys and records the current set of pressed keys to an array. As long as the array is not full, new keys can be pressed and registered, however if the array is filled up, caused by too many simultaneous keys pressed down at the same time, the keyboard will no longer register any new

key-presses beyond what is currently in the array. Otherwise, the key-press is detected and added to the array, and sent back to the host.

The `USB_HIDKeyPressUp` function works in a similar way, checking the array for keys that are not currently pressed, and making sure to clear out any key that has just been released from the array. After successfully clearing out any released keys, the array is sent to the host.

To handle the sending of reports back to the host, the `USB_HIDKeyboardReportInSend` function streamlines the process to replace the Get_Report request. It processes the current report and sends to the host, unless the pipeline is busy, in which results in the function setting up the report as the next in line for transmission.

3.6 CDC Overview

The USB Stack's Communication Device Class (CDC) implementation allows the user to perform a variety of networking and communication operations, and it is possible to extend the compatibility by writing drivers to support devices such as digital telephones or cable modems through the usage of the USB stack. With the ability to function as a virtual COM port, with an easy-to-use function set, CDC can be preferable to other standards. The internal USB and CDC solution can replace external USART → CDC translation hardware when implemented.

The USB CDC capabilities allows for certain benefits, such as:

- Can appear as a virtual COM port on PC-side applications
- Reliable data transfer through protocol-level error handling, data buffering, automatic flow control etc.
- Supports Abstract Control Model (ACM) functionality
- Supports multiple OS, including Windows, macOS and Linux
- Hardware indiscriminate protocol

3.7 USB CDC - How to use

The USB CDC class implementation handles basic CDC requests and mainly consists of a handler function and several helper functions to access status variables transmitted to and from the host.

The initialization of the CDC class is done by setting up a pointer to the USB descriptor and by registering the request handler in the USB core as a callback to be called when needed.

The `USB_CDCRequestHandler` function handles all requests not on the control endpoint, it starts by checking that the incoming request is meant for an interface. It then handles the request based on direction and type. If the request is not handled it will return "UNSUPPORTED".

`controlLineState` is a 16-bit variable that contains two status bits:

- Request To Send (RTS) - Data Circuit-terminating Equipment (DCE) code to indicate that the host request data out transmission
- Data Terminal Ready (DTR) - Data Terminal Equipment (DTE) code to indicate terminal window opened by host

The host sets the DTR bit through the request handler and it can be accessed using `USB_CDCDataTerminalReady`.

`lineCoding` is a struct containing information on the formatting for each character transmission and has the following members:

- Data Terminal Rate (DTERate) - Transmission rate in bits per second
- Character Format - The number of stop bits {1, 1.5, 2}
- Parity Type - Type of parity used {none, odd, even, mark, space}
- Data Bits - Number of actual data bits in transmission {5, 6, 7, 8, 16}

These values can be set by both the host and device and communicating them is done through the request handler and can be accessed using the following helper functions:

- `USB_CDCSetBaud`
- `USB_CDCGetBaud`
- `USB_CDCSetStopBits`
- `USB_CDCGetStopBits`
- `USB_CDCSetParity`
- `USB_CDCGetParity`
- `USB_CDCSetDataBits`
- `USB_CDCGetDataBits`

3.8 CDC: Virtual Serial Port - How to use

The USB CDC Class Virtual Serial Port (VSP) implementation handles setup for serial communication over USB. It consists of a handler function and helper functions to access and transmit data.

The initialization of the Virtual Serial Port is done by calling the CDC Class initialization. No other initialization is needed. The virtual serial port is double buffered in terms of receiving data from the USB host. This is done because the host needs a buffer to store transmitted data and will overwrite any data that's there.

The `USB_CDCVirtualSerialPortHandler` function handles all Virtual Serial Port related tasks. It needs to run in the main loop of the program or be called by interrupts when needed to ensure that the buffers don't fill up resulting in loss of incoming data.

The handler takes care of both in- and outgoing transactions and deals with outgoing transactions as follows:

1. Checks if the transmit buffer is empty - Skips the rest if it is
2. Checks if the transmit pipe is busy - Skips the rest if it is
3. Transmits data stored in the transmit buffer and calls the `USB_CDCDataTransmitted` callback function

The incoming transactions are handled as follows:

1. Checks if the outgoing transactions succeeded - Skips the rest if not
2. Checks if there is enough room in buffer to receive maximum size transaction - Skips rest if not
3. Checks if the receive pipe is busy - Skips if it is
4. Receives data if it is available and writes it to a temporary buffer and calls the `USB_CDCDataReceived` callback function to handle the received data

The `USB_CDCDataReceived` callback function moves the received data from the temporary receive buffer and adds it to a circular buffer that's used for external communication. The `USB_CDCDataTransmitted` callback function resets the transmit buffer so that its ready for new data.

To access the received data, the `USB_CDCRead` function returns the next data byte from the receive buffer. The `USB_CDCWrite` function adds one provided byte to the transmit buffer. The `CDC_TxBufferFull` function returns true if the transmit buffer is full and can't handle more data. These functions and the `USB_CDCVirtualSerialPortHandler` function use functions from the circular buffer implementation to add and retrieve data from the buffers.

The circular buffers are implemented to allow the USART and CDC to transmit data without having to worry about losing data. The `CIRCBUF_Enqueue` function adds data to buffer if the buffer is not full and returns the status of the action. The `CIRCBUF_Dequeue` function pulls one byte from the buffer as long as there is data in the buffer and returns the status of the action. The buffer

implementation also has some helper functions: `CIRCBUF_Empty` and `CIRCBUF_Full` check if the buffer is empty or full while `CIRCBUF_FreeSpace` returns the number of empty slots in the buffer. A new buffer is created by creating an `uint8_t` array of the size of the resulting buffer and then using the `CIRCULAR_BUFFER_t` type defines to create a circular buffer using the previously created array:

```
STATIC uint8_t bufferArray[BUFFER_SIZE];
STATIC CIRCULAR_BUFFER_t circularBuffer = {
    .content = bufferArray,
    .head = 0,
    .tail = 0,
    .maxLength = BUFFER_SIZE
};
```

3.9 Vendor Overview

For situations where the standard USB device classes do not meet the requirements of the application, the USB Device Stack also provides a starting point for developing vendor specific class applications. As per the USB specifications the vendor class is a base class defined for vendors to use as they please. That means the device does not conform to the pre-defined specifications of the HID, CDC or any of the base classes defined by the USB Implementers Forum. Instead, the developer defines custom specifications that might include protocols and device behavior.

The only requirement is that the USB device identifies itself to the USB host as a vendor class device so the host lets the device enumerate on the USB bus. The USB device identification process is included in the USB Device Stack's API. It is important to remember that a result of the Vendor Specific Class being completely customizable is that the host device might also require custom software or drivers to function.

3.10 Vendor - How to use

This section guides the user to set up and use the vendor class on the Microchip® USB Device Stack. It includes example usage of functions to implement the vendor class on a USB device.

The stack includes functions to help set up and use the vendor class following the provided guidelines. Using these is highly recommended, to avoid unexpected behavior when using Microchip® devices. The vendor class supports control, interrupt, isochronous and bulk transfer, allowing for versatility.

Before starting on a USB vendor project, consider if any of the other classes meet the requirement for the application. In most cases, it is better to use a specific USB class rather than a default vendor class. If none of the other standards meet the requirements, the vendor class implementation might be considered the best option.

The fastest way to implement the vendor class on any device is to use MCC Melody to set up the USB vendor class for any project. This can be done in a few steps:

- Create a new project in MPLAB® X IDE
- Import the USB Device Stack from the UI
- From the USB Device Tab, Select the USB vendor class
- Configure the vendor class settings according to project requirements if necessary
- Click “Generate”

It is also possible to enable the vendor class manually by creating your own project and adding the necessary code. This however requires more work than the MCC Melody code generation.

When writing the project code manually, it is important to note that the USB descriptor is an important part of USB communication, and needs to be set up correctly for the USB Device Stack to operate properly. For the vendor class, it is necessary to set the USB descriptors ID to `0xFF` to signal a vendor class being used, which is done by setting the structure`USB_DEVICE_DESCRIPTOR_t`’s

`bDeviceClass` value. This notifies the host system about the device's vendor-specific class, prompting it to refer to the corresponding vendor-specific drivers for device communication.

The next step is to communicate to the stack that the Vendor class is used. The USB stack includes a `USB_VendorClassInit()` helper function to easily initialize the class for the user, with the three input arguments, each one a callback to functions for enabling the interface, vendor control function and for disabling the interface. Each of these three callback functions might be necessary to set up, depending on which callbacks will be necessary in the code.

Upon successful initialization, the generic function driver interface activates and configures all attached endpoints. These are primed and set ready for data transfers upon receipt of the `USB_DEVICE_EVENT_CONFIGURED` event.

Note that unlike standard USB class drivers, the device layer does not automatically manage endpoints for vendor interface. This means that the application must maintain all active endpoints belonging to that interface in the correct configuration (e.g., bulk, interrupt). This involves enabling the endpoints for the desired transfer type according to pending transfers, and disabling them when a USB reset is received or when the configuration is changed.

4. USB Device Stack API Documentation

The following chapters contain API documentation autogenerated from the USB Device Stack code using doxygen.

4.1 Project MISRA C:2012 deviations

This page contains all project deviations, defined as deviations from MISRA C guidelines in a particular class of circumstances, for all files in the USB Device Stack for 8-bit AVR® MCUs.

Advisory: misra-c2012-2.5

Justification: False positive - This deviation explicitly only concerns the false positive on include guard macros. While rule 2.5 does not explicitly specify an exception for include guards, it is deemed that the wording of the rule effectively confirms that this is a false positive. The standard does not specify what "unused" means. An include guard macro is deemed as used as it does, in fact, have a function when used as an include guard. Moreover, the rationale for rule 2.5 states: "If a macro is declared but not used, then it is unclear to the reviewer if the macro is redundant or has been left unused by mistake". It is assumed that no code reviewer would think it is unclear whether or not an include guard is redundant. Finally, MISRA C:2012 Directive 4.10 states that: "Precautions shall be taken in order to prevent the contents of a header file being included more than once" and shows the usage of include guards as an example of compliance. Thus, the false positive can be suppressed for all include guards that trigger this false positive, even if the header-file is only included in one place. The latter reasoning is added because one cannot always know in advance whether or not a header file will be included more than once and include guards should thus be added regardless. Also, see thread-1317 on the official MISRA forum webpage for further confirmation that this is a false positive.

Include guard example:

```
#ifndef FOO_H
// cppcheck-suppress misra-c2012-2.5
#define FOO_H
//contents of file
#endif //FOO_H
```

4.2 Specific MISRA C:2012 deviations

This page contains all Specific MISRA C:2012 deviations.

Note:
Any project deviations are documented in Project MISRA C:2012 deviations

.

Global USB_DESCRIPTOR_PTR_t

Advisory: misra-c2012-19.2

Justification: Needed for the stack to parse through the configuration descriptors without pointer casting between the different descriptor types and uint8_t.

Global USB_EndpointInBufferSet **(uint8_t endpointAddress, uint8_t *bufAddress)**

Advisory: misra-c2012- 11.4

Justification: A conversion should not be performed between a pointer to object and an integer type. The EP.IN.DATAPTR register is a 16-bit register, expecting an AVR DU specific 16-bit RAM address.

Global USB_EndpointOutBufferSet **(uint8_t endpointAddress, uint8_t *bufAddress)**

Advisory: misra-c2012- 11.4

Justification: A conversion should not be performed between a pointer to object and an integer type. The EP.OUT.DATAPTR register is a 16-bit register, expecting an AVR DU specific 16-bit RAM address.

4.3 Supported MISRA C:2012 Rules

4.3.1 Supported MISRA C:2012 Rules

The USB Device Stack library is checked for any deviations from the MISRA C:2012 guidelines using the MISRA CLI MPLAB®X (MisraCheck plugin) static analyzer bundled with the MPLAB®X IDE version used during release verification (see the library release notes for this library version). The below table lists which of the MISRA rules are supported by the tool. The fields used in the table are defined as:

- **Rule ID** - The ID of the rule or directive being checked. These IDs correspond to the rule or directive numbers used in the MISRA C:2012 guidelines.
- **Classification** - Each MISRA C:2012 rule or directive is assigned one of three possible categories:
  - Mandatory
  - Required
  - Advisory
- **Status** - Describes which rules are enabled/disabled during verification or if a rule or directive is not supported by the tool

Rule ID	Classification	Status
1.1	Required	Not supported
1.2	Advisory	Not supported
1.3	Required	Enabled
1.4	Required	Enabled
2.1	Required	Enabled
2.2	Required	Enabled
2.3	Advisory	Disabled
2.4	Advisory	Disabled
2.5	Advisory	Disabled
2.6	Advisory	Enabled
2.7	Advisory	Enabled
3.1	Required	Enabled
3.2	Required	Enabled
4.1	Required	Enabled
4.2	Advisory	Enabled
4.3	Required	Enabled
4.4	Advisory	Enabled
4.5	Advisory	Enabled
4.6	Advisory	Enabled
4.7	Required	Enabled
4.8	Advisory	Enabled
4.9	Advisory	Enabled
4.10	Required	Enabled
4.11	Required	Enabled
4.12	Required	Enabled
4.13	Advisory	Enabled

.....continued

Rule ID	Classification	Status
4.14	Required	Enabled
5.1	Required	Enabled
5.2	Required	Enabled
5.3	Required	Enabled
5.4	Required	Enabled
5.5	Required	Enabled
5.6	Required	Enabled
5.7	Required	Enabled
5.8	Required	Enabled
5.9	Advisory	Enabled
6.1	Required	Enabled
6.2	Required	Enabled
7.1	Required	Enabled
7.2	Required	Enabled
7.3	Required	Enabled
7.4	Required	Enabled
8.1	Required	Enabled
8.2	Required	Enabled
8.3	Required	Enabled
8.4	Required	Disabled
8.5	Required	Enabled
8.6	Required	Enabled
8.7	Advisory	Disabled
8.8	Required	Enabled
8.9	Advisory	Enabled
8.10	Required	Enabled
8.11	Advisory	Enabled
8.12	Required	Enabled
8.13	Advisory	Enabled
8.14	Required	Enabled
9.1	Mandatory	Enabled
9.2	Required	Enabled
9.3	Required	Enabled
9.4	Required	Enabled
9.5	Required	Enabled
10.1	Required	Enabled
10.2	Required	Enabled
10.3	Required	Enabled
10.4	Required	Enabled
10.5	Advisory	Enabled

.....continued

Rule ID	Classification	Status
10.6	Required	Enabled
10.7	Required	Enabled
10.8	Required	Enabled
11.1	Required	Enabled
11.2	Required	Enabled
11.3	Required	Enabled
11.4	Advisory	Enabled
11.5	Advisory	Enabled
11.6	Required	Enabled
11.7	Required	Enabled
11.8	Required	Enabled
11.9	Required	Enabled
12.1	Advisory	Enabled
12.2	Required	Enabled
12.3	Advisory	Enabled
12.5	Mandatory	Enabled
12.4	Advisory	Enabled
13.1	Required	Enabled
13.2	Required	Enabled
13.3	Advisory	Enabled
13.4	Advisory	Enabled
13.5	Required	Enabled
13.6	Mandatory	Enabled
14.1	Required	Enabled
14.2	Required	Enabled
14.3	Required	Enabled
14.4	Required	Enabled
15.1	Advisory	Enabled
15.2	Required	Enabled
15.3	Required	Enabled
15.4	Advisory	Enabled
15.5	Advisory	Enabled
15.6	Required	Enabled
15.7	Required	Enabled
16.1	Required	Enabled
16.2	Required	Enabled
16.3	Required	Enabled
16.4	Required	Enabled
16.5	Required	Enabled
16.6	Required	Enabled

.....continued

Rule ID	Classification	Status
16.7	Required	Enabled
17.1	Required	Enabled
17.2	Required	Enabled
17.3	Mandatory	Not supported
17.4	Mandatory	Enabled
17.5	Advisory	Enabled
17.6	Mandatory	Enabled
17.7	Required	Enabled
17.8	Advisory	Enabled
18.1	Required	Enabled
18.2	Required	Enabled
18.3	Required	Enabled
18.4	Advisory	Enabled
18.5	Advisory	Enabled
18.6	Required	Enabled
18.7	Required	Enabled
18.8	Required	Enabled
19.1	Mandatory	Enabled
19.2	Advisory	Enabled
20.1	Advisory	Enabled
20.2	Required	Enabled
20.3	Required	Enabled
20.4	Required	Enabled
20.5	Advisory	Enabled
20.6	Required	Enabled
20.7	Required	Enabled
20.8	Required	Enabled
20.9	Required	Enabled
20.10	Advisory	Enabled
20.11	Required	Enabled
20.12	Required	Enabled
20.13	Required	Enabled
20.14	Required	Enabled
21.1	Required	Enabled
21.2	Required	Enabled
21.3	Required	Enabled
21.4	Required	Enabled
21.5	Required	Enabled
21.6	Required	Enabled
21.7	Required	Enabled

.....continued		
Rule ID	**Classification**	**Status**
21.8	Required	Enabled
21.9	Required	Enabled
21.10	Required	Enabled
21.11	Required	Enabled
21.12	Required	Enabled
21.13	Mandatory	Enabled
21.14	Required	Enabled
21.15	Required	Enabled
21.16	Required	Enabled
21.17	Mandatory	Enabled
21.18	Mandatory	Enabled
21.19	Mandatory	Enabled
21.20	Mandatory	Enabled
21.21	Required	Enabled
22.1	Required	Enabled
22.2	Mandatory	Enabled
22.3	Required	Enabled
22.4	Mandatory	Enabled
22.5	Mandatory	Enabled
22.6	Mandatory	Enabled
22.7	Required	Enabled
22.8	Required	Enabled
22.9	Required	Enabled
22.10	Required	Enabled

4.4 Module Documentation

4.4.1 USB Communications Device Class (CDC)

This file contains prototypes and data types for a CDC application.

4.4.1.1 Module description

This file contains prototypes and data types for a CDC application.

Version: USB Device Stack Driver Version 1.0.0

4.4.1.1.1 Data structures

- struct CIRCULAR_BUFFER_t
  Type define for circular buffers of varying length.
- struct USB_CDC_HEADER_FUNCTIONAL_DESCRIPTOR_t
  Type define for the CDC Header functional descriptor subtype.
- struct USB_CDC_ACM_FUNCTIONAL_DESCRIPTOR_t
  Type define for the CDC Abstract Control Management functional descriptor subtype.
- struct USB_CDC_COUNTRY_SELECTION_FUNCTIONAL_DESCRIPTOR_t
  Type define for the CDC country selection functional descriptor subtype.

- struct USB_CDC_LINE_CODING_t
  Type define for CDC Line Encoding.

4.4.1.1.2 Functions

- BUFFER_RETURN_CODE_t CIRCBUF_Enqueue (CIRCULAR_BUFFER_t *buffer, uint8_t data)
  Adds input data to circular buffer if there is space available.
- BUFFER_RETURN_CODE_t CIRCBUF_Dequeue (CIRCULAR_BUFFER_t *buffer, uint8_t *data)
  Pulls data from the circular buffer if it's available.
- bool CIRCBUF_Empty (CIRCULAR_BUFFER_t *buffer)
  Checks if the circular buffer is empty.
- bool CIRCBUF_Full (CIRCULAR_BUFFER_t *buffer)
  Checks if the circular buffer is full.
- uint16_t CIRCBUF_FreeSpace (CIRCULAR_BUFFER_t *buffer)
  Returns the number of available bytes in the circular buffer.
- void USB_CDCInitialize (void)
  Initializes the CDC class.
- RETURN_CODE_t USB_CDCRequestHandler (USB_SETUP_REQUEST_t *setupRequestPtr)
  Performs handling of control transfers.
- bool USB_CDCDataTerminalReady (void)
  Checks if the Data Terminal Equipment bit has been set from the host.
- void USB_CDCSetBaud (uint16_t baud)
  Sets the data transfer baud rate for the CDC communication.
- uint32_t USB_CDCGetBaud (void)
  Gets the data transfer baud rate for the CDC communication.
- void USB_CDCSetStopBits (USB_CDC_LINE_CODING_STOP_BITS_t numStopBits)
  Sets the number of data transfer stop bits for the CDC communication.
- USB_CDC_LINE_CODING_STOP_BITS_t USB_CDCGetStopBits (void)
  Gets the number of data transfer stop bits for the CDC communication.
- void USB_CDCSetParity (USD_CDC_LINE_CODING_PARITY_t parity)
  Sets the data transfer parity for the CDC communication.
- USD_CDC_LINE_CODING_PARITY_t USB_CDCGetParity (void)
  Gets the data transfer parity for the CDC communication.
- void USB_CDCSetDataBits (USD_CDC_LINE_CODING_DATA_BITS_t numDataBits)
  Sets the number of data transfer data bits for the CDC communication.
- USD_CDC_LINE_CODING_DATA_BITS_t USB_CDCGetDataBits (void)
  Gets the number of data transfer data bits for the CDC communication.
- void USB_CDCVirtualSerialPortInitialize (void)
  Initializes the CDC Virtual Serial Port functionality.
- RETURN_CODE_t USB_CDCVirtualSerialPortHandler (void)
  Performs Virtual Serial Port writes using the CDC class.
- CDC_RETURN_CODE_t USB_CDCRead (uint8_t *data)
  Pulls data from the CDC receive buffer.
- CDC_RETURN_CODE_t USB_CDCWrite (uint8_t data)

Adds data to the CDC transmit buffer.

- bool USB_CDCTxBusy (void)
  Checks if the transmit buffer is full.
- void USB_CDCDataTransmitted (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)
  Callback function called after the USB IN transaction started.
- void USB_CDCDataReceived (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)
  Callback function called after the USB OUT transaction started.

4.4.1.2 Function Documentation

4.4.1.2.1 CIRCBUF_Dequeue()

BUFFER_RETURN_CODE_t CIRCBUF_Dequeue (CIRCULAR_BUFFER_t * buffer, uint8_t * data)

Pulls data from the circular buffer if it's available.

Parameters:

buffer	- Circular buffer address
data	- Output data variable address

Returns:

status - Result of the retrieval process

4.4.1.2.2 CIRCBUF_Empty()

bool CIRCBUF_Empty (CIRCULAR_BUFFER_t * buffer)

Checks if the circular buffer is empty.

Parameters:

buffer	- Circular buffer address

Return values:

0	- Buffer not full
1	- Buffer full

4.4.1.2.3 CIRCBUF_Enqueue()

BUFFER_RETURN_CODE_t CIRCBUF_Enqueue (CIRCULAR_BUFFER_t * buffer, uint8_t data)

Adds input data to circular buffer if there is space available.

Parameters:

buffer	- Circular buffer address
data	- Intput data

Returns:

status - Result of the addition process

4.4.1.2.4 CIRCBUF_FreeSpace()

uint16_t CIRCBUF_FreeSpace (CIRCULAR_BUFFER_t * buffer)

Returns the number of available bytes in the circular buffer.

Parameters:

buffer	- Circular buffer address

Returns:

freeSpace - Available bytes

4.4.1.2.5 CIRCBUF_Full()

bool CIRCBUF_Full (CIRCULAR_BUFFER_t * buffer)

Checks if the circular buffer is full.

Parameters:

buffer	- Circular buffer address

Return values:

0	- Buffer not full
1	- Buffer full

4.4.1.2.6 USB_CDCDataReceived()

void USB_CDCDataReceived (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)

Callback function called after the USB OUT transaction started.

Parameters:

pipe	- USB pipe used for the started transaction
status	- Transfer status
bytesTransferred	- Number of bytes received in the transaction

Returns:

None.

4.4.1.2.7 USB_CDCDataTerminalReady()

bool USB_CDCDataTerminalReady (void)

Checks if the Data Terminal Equipment bit has been set from the host.

Parameters:

None.

Return values:

0	- False if bit not set
1	- True if bit set

4.4.1.2.8 USB_CDCDataTransmitted()

void USB_CDCDataTransmitted (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)

Callback function called after the USB IN transaction started.

Parameters:

pipe	- USB pipe used for the started transaction
status	- Transfer status

bytesTransferred	- Number of bytes transmitted in the transaction

Returns:

None.

4.4.1.2.9 USB_CDCGetBaud()

uint32_t USB_CDCGetBaud (void)

Gets the data transfer baud rate for the CDC communication.

Parameters:

None.

Returns:

baud - Data transfer baud rate

4.4.1.2.10 USB_CDCGetDataBits()

USD_CDC_LINE_CODING_DATA_BITS_t USB_CDCGetDataBits (void)

Gets the number of data transfer data bits for the CDC communication.

Parameters:

None.

Returns:

numDataBits - Number of data bits

4.4.1.2.11 USB_CDCGetParity()

USD_CDC_LINE_CODING_PARITY_t USB_CDCGetParity (void)

Gets the data transfer parity for the CDC communication.

Parameters:

None.

Returns:

parity - Data transfer parity

4.4.1.2.12 USB_CDCGetStopBits()

USB_CDC_LINE_CODING_STOP_BITS_t USB_CDCGetStopBits (void)

Gets the number of data transfer stop bits for the CDC communication.

Parameters:

None.

Returns:

numStopBits - Number of stop bits

4.4.1.2.13 USB_CDCInitialize()

void USB_CDCInitialize (void)

Initializes the CDC class.

Parameters:

None.

Returns:

None.

4.4.1.2.14 USB_CDCRead()

CDC_RETURN_CODE_t USB_CDCRead (uint8_t * data)

Pulls data from the CDC receive buffer.

Parameters:

buffer	- Pointer to application receive buffer

Returns:

status - Result of the called circular buffer function

4.4.1.2.15 USB_CDCRequestHandler()

RETURN_CODE_t USB_CDCRequestHandler (USB_SETUP_REQUEST_t * setupRequestPtr)

Performs handling of control transfers.

Parameters:

setupRequestPtr	- Pointer to the Setup Request struct

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.1.2.16 USB_CDCSetBaud()

void USB_CDCSetBaud (uint16_t baud)

Sets the data transfer baud rate for the CDC communication.

Parameters:

baud	- Data transfer baud rate

Returns:

None.

4.4.1.2.17 USB_CDCSetDataBits()

void USB_CDCSetDataBits (USD_CDC_LINE_CODING_DATA_BITS_t numDataBits)

Sets the number of data transfer data bits for the CDC communication.

Parameters:

numDataBits	- Number of data bits

Returns:

None.

4.4.1.2.18 USB_CDCSetParity()

void USB_CDCSetParity (USD_CDC_LINE_CODING_PARITY_t parity)

Sets the data transfer parity for the CDC communication.

Parameters:

parity	- Data transfer parity

Returns:

None.

4.4.1.2.19 USB_CDCSetStopBits()

void USB_CDCSetStopBits (USB_CDC_LINE_CODING_STOP_BITS_t numStopBits)

Sets the number of data transfer stop bits for the CDC communication.

Parameters:

numStopBits	- Number of stop bits

Returns:

None.

4.4.1.2.20 USB_CDCTxBusy()

bool USB_CDCTxBusy (void)

Checks if the transmit buffer is full.

Parameters:

None.

Return values:

0	- Buffer not full
1	- Buffer full

4.4.1.2.21 USB_CDCVirtualSerialPortHandler()

RETURN_CODE_t USB_CDCVirtualSerialPortHandler (void)

Performs Virtual Serial Port writes using the CDC class.

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.1.2.22 USB_CDCVirtualSerialPortInitialize()

void USB_CDCVirtualSerialPortInitialize (void)

Initializes the CDC Virtual Serial Port functionality.

Parameters:

None.

Returns:

None.

4.4.1.2.23 USB_CDCWrite()

CDC_RETURN_CODE_t USB_CDCWrite (uint8_t data)

Adds data to the CDC transmit buffer.

Parameters:

data	- Pointer to data to be transmitted

length	- Length in number of bytes for data to be transmitted

Returns:

status - Result of the called circular buffer function

4.4.2 USB Common Elements

Common elements for the USB stack.

4.4.2.1 Module description

Common elements for the USB stack.

Version: USB Common Elements Version 1.0.0

4.4.2.1.1 Definitions

- #define MAX_ENDPOINT_SIZE_DEFAULT (64)
  The maximum endpoint packet size for the default endpoint types (control, bulk, interrupt).
- #define MAX_ENDPOINT_SIZE_ISO (1023)
  The maximum endpoint packet size for the isochronous endpoint type.

4.4.2.1.2 Enumerations

- enum RETURN_CODE_enum { UNSUPPORTED = 2, UNINITIALIZED = 1, SUCCESS = 0, ENDPOINT_SIZE_ERROR = -1, ENDPOINT_ADDRESS_ERROR = -2, ENDPOINT_DIRECTION_ERROR = -3, ENDPOINT_TYPE_ERROR = -4, ENDPOINT_AZLP_ERROR = -5, ENDPOINT_ALIGN_ERROR = -6, PIPE_BUSY_ERROR = -10, PIPE_TRANSFER_ERROR = -11, CONTROL_SIZE_ERROR = -20, CONTROL_TRANSACTION_STATUS_ERROR = -21, CONTROL_SETUP_CALLBACK_ERROR = -22, CONTROL_SETUP_DIRECTION_ERROR = -23, DESCRIPTOR_POINTER_ERROR = -30, DESCRIPTOR_CONFIGURATIONS_ERROR = -31, DESCRIPTOR_INTERFACE_ERROR = -32, DESCRIPTOR_ENDPOINT_ERROR = -33, DESCRIPTOR_REQUEST_ERROR = -34, DESCRIPTOR_SEARCH_ERROR = -35, INTERFACE_SET_ERROR = -40, INTERFACE_GET_ERROR = -41, USB_CONNECTION_ERROR = -50, USB_CLASS_ERROR = -60 }
  Describes the different function return reserved codes used by the USB stack.

4.4.2.2 Definition Documentation

4.4.2.2.1 MAX_ENDPOINT_SIZE_DEFAULT

#define MAX_ENDPOINT_SIZE_DEFAULT (64)

The maximum endpoint packet size for the default endpoint types (control, bulk, interrupt).

4.4.2.2.2 MAX_ENDPOINT_SIZE_ISO

#define MAX_ENDPOINT_SIZE_ISO (1023)

The maximum endpoint packet size for the isochronous endpoint type.

4.4.2.3 Enumeration Type Documentation

4.4.2.3.1 RETURN_CODE_enum

enum RETURN_CODE_enum

Describes the different function return reserved codes used by the USB stack.

UNSUPPORTED	Action not supported by the USB Device Stack
UNINITIALIZED	Status unchanged since initialization
SUCCESS	Action successfully executed
ENDPOINT_SIZE_ERROR	Error related to the wMaxPacketSize in the endpoint configuration

ENDPOINT_ADDRESS_ERROR	Error related to the address byte of bEndpointAddress in the endpoint configuration
ENDPOINT_DIRECTION_ERROR	Error related to the direction byte of bEndpointAddress in the endpoint configuration
ENDPOINT_TYPE_ERROR	Error related to the type bitmask of bmAttributes in the endpoint configuration
ENDPOINT_AZLP_ERROR	Error related to the Auto Zero Length Packet (AZLP) setting in the endpoint configuration
ENDPOINT_ALIGN_ERROR	Error related to unaligned OUT transactions when using multipacket
PIPE_BUSY_ERROR	Error triggered by the pipe being busy while attempting a read or write transaction
PIPE_TRANSFER_ERROR	Error triggered by a failed pipe transaction
CONTROL_SIZE_ERROR	Error related to the size of the control endpoint transaction packet
CONTROL_TRANSACTION_STATUS_ERROR	Error triggered by a failed transaction on the control endpoint
CONTROL_SETUP_CALLBACK_ERROR	Error triggered by a failed control setup
CONTROL_SETUP_DIRECTION_ERROR	Error triggerd by a control setup IN request not requesting any data
DESCRIPTOR_POINTER_ERROR	Error triggered by an invalid descriptor pointer
DESCRIPTOR_CONFIGURATIONS_ERROR	Error triggered by an invalid configuration descriptor pointer
DESCRIPTOR_INTERFACE_ERROR	Error triggered by an invalid interface descriptor pointer
DESCRIPTOR_ENDPOINT_ERROR	Error triggered by an invalid endpoint descriptor pointer
DESCRIPTOR_REQUEST_ERROR	Error triggered by a failed attempt at retrieving a descriptor pointer
DESCRIPTOR_SEARCH_ERROR	Error triggered by an incorrect descriptor structure
INTERFACE_SET_ERROR	Error triggered by a failure to set an interface
INTERFACE_GET_ERROR	Error triggered by a failure to retrieve an interface
USB_CONNECTION_ERROR	Error triggered by a failure during setup device request
USB_CLASS_ERROR	Error triggered by an invalid class code

4.4.3 USB Core Layer

Core functionality for the USB stack.

4.4.3.1 Module description

Core functionality for the USB stack.

Version: USB Device Core Version 1.0.0

4.4.3.1.1 Modules

- USB Common Elements
  Common elements for the USB stack.
- USB Descriptor Definitions
  In this file the active configuration and interfaces can be changed. The active configuration and active interface is referenced by a pointer as two global variables.
- USB Core Events
  Event handling for the USB Core Stack.
- USB Core Requests
  USB Device Core Requests handling.
- USB Core Transfer
  USB core layer implementation file.
- USB Protocol

Common data structures, enumerations and macro definitons based on the USB 2.0 base protocol.

4.4.3.1.2 Data structures

- struct USB_EVENT_HANDLERS_struct

  Represents the event callbacks, the configuration and the enumeration setups.

4.4.3.1.3 Functions

- RETURN_CODE_t USB_SetupProcess (USB_SETUP_REQUEST_t *setupRequestPtr)

  Setup function for the Standard Device Request USB 2.0 Specification Ch 9.4.
- RETURN_CODE_t USB_Start (void)

  Starts the USB peripheral, configures the callbacks and attaches it to the bus.
- RETURN_CODE_t USB_Stop (void)

  Stops the USB peripheral and detaches it from the bus.
- RETURN_CODE_t USB_Reset (void)

  Resets the USB peripheral.
- RETURN_CODE_t USB_DescriptorPointersSet (USB_DESCRIPTOR_POINTERS_t *descriptorPtr)

  Handles Descriptor pointer setup. Sets the address to the application descriptor pointers. Checks if the device pointer and a pointer to the start of the application configuration(s) are set before saving the address to the USB Core Stack.
- RETURN_CODE_t USB_DescriptorConfigurationEnable (uint8_t configurationValue)

  Enables endpoint configuration descriptor.
- bool USB_DescriptorActiveConfigurationSelfPoweredGet (void)

  Gets the Self-Powered setting from the active configuration.
- bool USB_DescriptorActiveConfigurationRemoteWakeupGet (void)

  Gets the Remote Wake-up setting from the active configuration.
- uint8_t USB_DescriptorActiveConfigurationValueGet (void)

  Gets the active configuration value.
- RETURN_CODE_t USB_DescriptorInterfaceConfigure (uint8_t interfaceNumber, uint8_t alternateSetting, bool enable)

  Enables or Disables an Interface Descriptor.
- RETURN_CODE_t USB_DescriptorPointerGet (USB_DESCRIPTOR_TYPE_t descriptor, uint8_t attribute, uint8_t **descriptorPtr, uint16_t *descriptorLength)

  Gets the pointer to the descriptor.
- RETURN_CODE_t USB_DescriptorStringPointerGet (uint8_t stringIndex, uint16_t langID, uint8_t **descriptorAddressPtr, uint16_t *descriptorLength)

  Gets the pointer to the string descriptor.
- RETURN_CODE_t ActiveAlternateSettingGet (uint8_t interfaceNumber, uint8_t *alternateSetting)

  Get the active alternate interface number for an interface.
- RETURN_CODE_t ActiveAlternateSettingSet (uint8_t interfaceNumber, uint8_t alternateSetting)

  Set the active alternate interface number for an interface.
- RETURN_CODE_t ConfigurationPointerGet (uint8_t configurationValue, USB_CONFIGURATION_DESCRIPTOR_t **configurationPtr)

  Collects the configuration pointer.
- RETURN_CODE_t DescriptorEndpointsConfigure (USB_INTERFACE_DESCRIPTOR_t *interfacePtr, bool enable)

Configures the endpoints as given in the descriptor.

- RETURN_CODE_t NextDescriptorPointerGet (USB_DESCRIPTOR_TYPE_t descriptorType, USB_DESCRIPTOR_HEADER_t **descriptorHeaderPtr)
  Gets the next descriptor.
- RETURN_CODE_t USB_EventHandler (void)
  Handles the different types of events.
- void USB_SetConfigurationCallbackRegister (USB_SETUP_EVENT_CALLBACK_t callback)
  Registers a callback for Set Configuration requests.
- void USB_SetInterfaceCallbackRegister (USB_SETUP_EVENT_CALLBACK_t callback)
  Registers a callback for Set Interface requests.
- void USB_InterfaceDisabledCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for disabling interfaces.
- void USB_VendorRequestCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)
  Registers a callback for vendor requests.
- void USB_ClassRequestCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)
  Registers a callback for class requests.
- void USB_OtherRequestCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)
  Registers a callback for other requests.
- void USB_SOFCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for Start-Of-Frame (SOF) events.
- void USB_ResetCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for Reset events.
- void USB_SuspendCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for Suspend events.
- void USB_ResumeCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for Resume events.

4.4.3.2 Function Documentation

4.4.3.2.1 ActiveAlternateSettingGet()

RETURN_CODE_t ActiveAlternateSettingGet (uint8_t interfaceNumber, uint8_t * alternateSetting)

Get the active alternate interface number for an interface.

Parameters:

interfaceNumber	- The interface number to get the alternate setting for
*alternateSetting	- The requested alternate setting

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.2 ActiveAlternateSettingSet()

RETURN_CODE_t ActiveAlternateSettingSet (uint8_t interfaceNumber, uint8_t alternateSetting)

Set the active alternate interface number for an interface.

Parameters:

interfaceNumber	- The interface number to set the alternate setting for

alternateSetting	- The alternate setting to set

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.3 ConfigurationPointerGet()

RETURN_CODE_t ConfigurationPointerGet (uint8_t configurationValue, USB_CONFIGURATION_DESCRIPTOR_t ** configurationPtr)

Collects the configuration pointer.

Parameters:

configurationValue	- Value of the referenced configuration
**configurationPtr	- Pointer to the configuration

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.4 DescriptorEndpointsConfigure()

RETURN_CODE_t DescriptorEndpointsConfigure (USB_INTERFACE_DESCRIPTOR_t * interfacePtr, bool enable)

Configures the endpoints as given in the descriptor.

Parameters:

*interfacePtr	- Pointer to an interface
enable	- Boolean to enable or disable the endpoint

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.5 NextDescriptorPointerGet()

RETURN_CODE_t NextDescriptorPointerGet (USB_DESCRIPTOR_TYPE_t descriptorType, USB_DESCRIPTOR_HEADER_t ** descriptorHeaderPtr)

Gets the next descriptor.

Parameters:

descriptorType	- Selected descriptor type
**descriptorHeaderPtr	- Pointer to the descriptor header

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.6 USB_ClassRequestCallbackRegister()

void USB_ClassRequestCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)

Registers a callback for class requests.

Parameters:

callback	- Reference for the callback function

Returns:

None.

4.4.3.2.7 USB_DescriptorActiveConfigurationRemoteWakeupGet()

bool USB_DescriptorActiveConfigurationRemoteWakeupGet (void)

Gets the Remote Wake-up setting from the active configuration.

Parameters:

None.

Return values:

0	- Remote Wake-up is not enabled
1	- Remote Wake-up is enabled

4.4.3.2.8 USB_DescriptorActiveConfigurationSelfPoweredGet()

bool USB_DescriptorActiveConfigurationSelfPoweredGet (void)

Gets the Self-Powered setting from the active configuration.

Parameters:

None.

Return values:

0	- Self-Powered is not enabled
1	- Self-Powered is enabled

4.4.3.2.9 USB_DescriptorActiveConfigurationValueGet()

uint8_t USB_DescriptorActiveConfigurationValueGet (void)

Gets the active configuration value.

Parameters:

None.

Returns:

The active configuration value

4.4.3.2.10 USB_DescriptorConfigurationEnable()

RETURN_CODE_t USB_DescriptorConfigurationEnable (uint8_t configurationValue)

Enables endpoint configuration descriptor.

The USB Device Enable Endpoint function, from USB 2.0 Specification Ch. 9.6.6.

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of this descriptor in bytes
1	bDescriptorType	1	Constant	CONFIGURATION Descriptor Type
2	bEndpointAddress	1	Endpoint	Address of the endpoint on the USB device described by this descriptor. Address is encoded by the following: Bit 3-0: The endpoint number, Bit 6-4: Reserved, reset to zero, Bit 7: Direction, ignored for control endpoint. 0 = OUT endpoint, 1 = IN endpoint
3	bmAttributes	1	Bitmap	Describes the endpoint attributes when it is configured using the bConfigurationValue. Bit 1-0: Transfer type, 00=Control, 01=Isochronous, 10=Bulk, 11=Interrupt. If not isochronous endpoint, bit 5-2 are reserved and must be set to zero. If isochronous: Bit 3-2: Synchronization type, 00 = No Sync, 01 = Async, 10 = Adaptive, 11 = Sync. Bit 5-4: Usage type, 00 = Data endpoint, 01 = Feedback endpoint, 10 = Implicit feedback Data endpoint, 11 = Reserved. All other bits are reserved and must be reset to zero. Reserved bits must be ignored by the host.

.....continued

Offset	Field	Size	Value	Description
4	wMaxPacketSize	2	Number	Maximum packet size this endpoint is capable of RX/TX when this configuration is selected. For isochronous endpoints, this value is used to reserve bus time in the schedule. For all endpoints, bit 10-0 specify maximum packet size. For high-speed isochronous and interrupt endpoints: Bit 12-11 number of additional transaction opportunities per microframe, 00 = None (1 transaction per microframe), 01 = 1 additional, 10 = 2 additional, 11 = Reserved. Bits 15-13 are reserved and must be set to zero.

Parameters:

configurationValue	- The value of the configuration to be enabled

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.11 USB_DescriptorInterfaceConfigure()

RETURN_CODE_t USB_DescriptorInterfaceConfigure (uint8_t interfaceNumber, uint8_t alternateSetting, bool enable)

Enables or Disables an Interface Descriptor.

The USB Device Enable Interface Descriptor, from USB 2.0 Specification Ch. 9.6.5.

Offset	Field	Size	Value	Description
2	bInterfaceNumber	1	Number	Number of this interface. Zero-based value identifying the index in the array of concurrent interfaces supported by this configuration
3	bAlternateSetting	1	Number	Value used to select this alternate setting for the interface identified in the prior field

Parameters:

interfaceNumber	- Interface number value
alternateSetting	- Alternative settings value, ignored if enable is false .
enable	- Enable or disable the interface.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.12 USB_DescriptorPointerGet()

RETURN_CODE_t USB_DescriptorPointerGet (USB_DESCRIPTOR_TYPE_t descriptor, uint8_t attribute, uint8_t ** descriptorPtr, uint16_t * descriptorLength)

Gets the pointer to the descriptor.

Parameters:

descriptor	- Descriptor type
attribute	- Attribute type
**descriptorPtr	- Pointer to the descriptor
*descriptorLength	- Length of the descriptor

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.13 USB_DescriptorPointersSet()

RETURN_CODE_t USB_DescriptorPointersSet (USB_DESCRIPTOR_POINTERS_t * descriptorPtr)

Handles Descriptor pointer setup. Sets the address to the application descriptor pointers. Checks if the device pointer and a pointer to the start of the application configuration(s) are set before saving the address to the USB Core Stack.

Parameters:

*descriptorPtr	- The address of the application descriptor pointers

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.14 USB_DescriptorStringPointerGet()

RETURN_CODE_t USB_DescriptorStringPointerGet (uint8_t stringIndex, uint16_t langID, uint8_t ** descriptorAddressPtr, uint16_t * descriptorLength)

Gets the pointer to the string descriptor.

Parameters:

stringIndex	- Index of the string
langID	- Language ID
**descriptorAddressPtr	- Pointer to the descriptor
*descriptorLength	- Length of the descriptor

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.15 USB_EventHandler()

RETURN_CODE_t USB_EventHandler (void)

Handles the different types of events.

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.16 USB_InterfaceDisabledCallbackRegister()

void USB_InterfaceDisabledCallbackRegister (USB_EVENT_CALLBACK_t callback)

Registers a callback for disabling interfaces.

Parameters:

callback	- Reference for the callback function

Returns:

None.

4.4.3.2.17 USB_OtherRequestCallbackRegister()

void USB_OtherRequestCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)

Registers a callback for other requests.

Parameters:

callback	- Reference for the callback function

Returns:

None.

4.4.3.2.18 USB_Reset()

RETURN_CODE_t USB_Reset (void)

Resets the USB peripheral.

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.19 USB_ResetCallbackRegister()

void USB_ResetCallbackRegister (USB_EVENT_CALLBACK_t callback)

Registers a callback for Reset events.

Parameters:

callback	- Reference for the callback function

Returns:

None.

4.4.3.2.20 USB_ResumeCallbackRegister()

void USB_ResumeCallbackRegister (USB_EVENT_CALLBACK_t callback)

Registers a callback for Resume events.

Parameters:

callback	- Reference for the callback function

Returns:

None.

4.4.3.2.21 USB_SetConfigurationCallbackRegister()

void USB_SetConfigurationCallbackRegister (USB_SETUP_EVENT_CALLBACK_t callback)

Registers a callback for Set Configuration requests.

Parameters:

callback	- Reference for the callback function

Returns:

None.

4.4.3.2.22 USB_SetInterfaceCallbackRegister()

void USB_SetInterfaceCallbackRegister (USB_SETUP_EVENT_CALLBACK_t callback)

Registers a callback for Set Interface requests.

Parameters:

callback	- Reference for the callback function

Returns:

None.

4.4.3.2.23 USB_SetupProcess()

RETURN_CODE_t USB_SetupProcess (USB_SETUP_REQUEST_t * setupRequestPtr)

Setup function for the Standard Device Request USB 2.0 Specification Ch 9.4.

bRequest	wValue	wIndex	wLength	Data
CLEAR_FEATURE	Feature Selector	Zero	None	
CLEAR_FEATURE	Feature Selector	Interface	None	
CLEAR_FEATURE	Feature Selector	Endpoint	None	
GET_CONFIGURATION	Zero	Zero	One	Configuration Value
GET_DESCRIPTOR	Descriptor type and Descriptor index	Zero or Language ID	Descriptor Length	Descriptor
GET_INTERFACE	Zero	Interface	One	Alternate Interface
GET_STATUS	Zero	Zero Interface Endpoint	Two	Device status
GET_STATUS	Zero	Interface	Two	Interface Status
GET_STATUS	Zero	Endpoint	Two	Endpoint Status
SET_ADDRESS	Device Address	Zero	Zero	None
SET_CONFIGURATION	Configuration Value	Zero	Zero	None
SET_DESCRIPTOR	Descriptor type and Descriptor index	Zero or Language ID	Descriptor Length	Descriptor
SET_FEATURE	Feature Selector	Zero Interface Endpoint	Zero	None
SET_FEATURE	Feature Selector	Interface	Zero	
SET_FEATURE	Feature Selector	Endpoint	Zero	
SET_INTERFACE	Alternate Setting	Interface	Zero	None
SYNCH_FRAME	Zero	Endpoint	Two	Frame Number

Parameters:

*setupRequestPtr	- Pointer to the setup request and its data

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.24 USB_SOFCallbackRegister()

void USB_SOFCallbackRegister (USB_EVENT_CALLBACK_t callback)

Registers a callback for Start-Of-Frame (SOF) events.

Parameters:

callback	- Reference for the callback function

Returns:

None.

4.4.3.2.25 USB_Start()

RETURN_CODE_t USB_Start (void)

Starts the USB peripheral, configures the callbacks and attaches it to the bus.

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.26 USB_Stop()

RETURN_CODE_t USB_Stop (void)

Stops the USB peripheral and detaches it from the bus.

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.3.2.27 USB_SuspendCallbackRegister()

void USB_SuspendCallbackRegister (USB_EVENT_CALLBACK_t callback)

Registers a callback for Suspend events.

Parameters:

callback	- Reference for the callback function

Returns:

None.

4.4.3.2.28 USB_VendorRequestCallbackRegister()

void USB_VendorRequestCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)

Registers a callback for vendor requests.

Parameters:

callback	- Reference for the callback function

Returns:

None.

4.4.4 USB Descriptor Definitions

In this file the active configuration and interfaces can be changed. The active configuration and active interface is referenced by a pointer as two global variables.

4.4.4.1 Module description

In this file the active configuration and interfaces can be changed. The active configuration and active interface is referenced by a pointer as two global variables.

Version: USB Device Core Version 1.0.0

4.4.4.1.1 Definitions

- #define USB_DEFAULT_INTERFACE 0u

  Default interface number.

- #define USB_DEFAULT_ALTERNATE_SETTING 0u
  Default alternate setting.
- #define USB_DESCRIPTOR_SEARCH_LIMIT 30u
  The number of descriptors NextDescriptorPointerGet will search through before returning an error.

4.4.4.2 Definition Documentation

4.4.4.2.1 USB_DEFAULT_ALTERNATE_SETTING

#define USB_DEFAULT_ALTERNATE_SETTING 0u

Default alternate setting.

4.4.4.2.2 USB_DEFAULT_INTERFACE

#define USB_DEFAULT_INTERFACE 0u

Default interface number.

4.4.4.2.3 USB_DESCRIPTOR_SEARCH_LIMIT

#define USB_DESCRIPTOR_SEARCH_LIMIT 30u

The number of descriptors NextDescriptorPointerGet will search through before returning an error.

4.4.5 USB Core Events

Event handling for the USB Core Stack.

4.4.5.1 Module description

Event handling for the USB Core Stack.

Version: USB Device Core Version 1.0.0

4.4.6 USB Core Requests

USB Device Core Requests handling.

4.4.6.1 Module description

USB Device Core Requests handling.

Version: USB Device Core Version 1.0.0

4.4.6.1.1 Definitions

- #define GET_STATUS_ENDPOINT_STALLED (1u << 0u)
  Mask for the endpoint stall status in the first byte of the data stage of the setup request.
- #define ENDPOINT_ADDRESS_MASK (0x7fu)
  Mask for the endpoint address in the wIndex field of the setup request.
- #define ENDPOINT_DIRECTION_BITPOSITION (7u)
  Bit position for the endpoint direction in the wIndex field of the setup request.
- #define GET_INTERFACE_REQUEST_NUMBER_MASK (0xffu)
  Mask for the interface number in the wIndex field of the setup request.
- #define GET_INTERFACE_REQUEST_WVALUE 0u
  Value for the wValue field of the setup request.
- #define GET_INTERFACE_RESPONSE_SIZE 1u
  Size of the response to the Get Interface request.

4.4.6.1.2 Functions

- RETURN_CODE_t USB_SetupProcessDeviceRequest (USB_SETUP_REQUEST_t *setupRequestPtr)

Setup function for the device requests.

- RETURN_CODE_t USB_SetupProcessEndpointRequest (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the endpoint requests.
- RETURN_CODE_t USB_SetupProcessInterfaceRequest (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the interface requests.
- RETURN_CODE_t SetupDeviceRequestGetStatus (void)
  Returns the status of the device features.
- RETURN_CODE_t SetupDeviceRequestClearFeature (USB_SETUP_REQUEST_t *setupRequestPtr)
  Clears the device feature.
- RETURN_CODE_t SetupDeviceRequestSetFeature (USB_SETUP_REQUEST_t *setupRequestPtr)
  Sets the device feature.
- RETURN_CODE_t SetupDeviceRequestSetAddress (uint8_t address)
  Sets the device address.
- void SetupDeviceAddressCallback (void)
  Callback function for the address.
- RETURN_CODE_t SetupDeviceRequestGetDescriptor (USB_SETUP_REQUEST_t *setupRequestPtr)
  Gets the device descriptor.
- RETURN_CODE_t SetupDeviceRequestGetConfiguration (void)
  Gets the device configuration.
- RETURN_CODE_t SetupDeviceRequestSetConfiguration (uint8_t configurationValue)
  Sets the device configuration.
- USB_PIPE_t EndpointFromRequestGet (uint16_t wIndex)
  Gets the endpoint status.
- RETURN_CODE_t SetupEndpointRequestGetStatus (USB_SETUP_REQUEST_t *setupRequestPtr)
  Gets the endpoint status.
- RETURN_CODE_t SetupEndpointRequestClearFeature (USB_SETUP_REQUEST_t *setupRequestPtr)
  Clears the endpoint feature.
- RETURN_CODE_t SetupEndpointRequestSetFeature (USB_SETUP_REQUEST_t *setupRequestPtr)
  Sets the endpoint feature.
- RETURN_CODE_t SetupEndpointRequestSynchFrame (void)
  Gets the current frame number.
- RETURN_CODE_t USB_SetupInterfaceRequestGetStatus (void)
  Returns status for the specified interface.
- RETURN_CODE_t USB_SetupInterfaceRequestGetInterface (USB_SETUP_REQUEST_t *setupRequestPtr)
  Returns the alternate setting for the specified interface.
- RETURN_CODE_t USB_SetupInterfaceRequestSetInterface (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the interface request to select the alternate setting.
- RETURN_CODE_t USB_SetupInterfaceRequestGetDescriptor (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the interface request for class-specific descriptors.

4.4.6.2 Definition Documentation

4.4.6.2.1 ENDPOINT_ADDRESS_MASK

#define ENDPOINT_ADDRESS_MASK (0x7fu)

Mask for the endpoint address in the wIndex field of the setup request.

4.4.6.2.2 ENDPOINT_DIRECTION_BITPOSITION

#define ENDPOINT_DIRECTION_BITPOSITION (7u)

Bit position for the endpoint direction in the wIndex field of the setup request.

4.4.6.2.3 GET_INTERFACE_REQUEST_NUMBER_MASK

#define GET_INTERFACE_REQUEST_NUMBER_MASK (0xffu)

Mask for the interface number in the wIndex field of the setup request.

4.4.6.2.4 GET_INTERFACE_REQUEST_WVALUE

#define GET_INTERFACE_REQUEST_WVALUE 0u

Value for the wValue field of the setup request.

4.4.6.2.5 GET_INTERFACE_RESPONSE_SIZE

#define GET_INTERFACE_RESPONSE_SIZE 1u

Size of the response to the Get Interface request.

4.4.6.2.6 GET_STATUS_ENDPOINT_STALLED

#define GET_STATUS_ENDPOINT_STALLED (1u << 0u)

Mask for the endpoint stall status in the first byte of the data stage of the setup request.

4.4.6.3 Function Documentation

4.4.6.3.1 EndpointFromRequestGet()

USB_PIPE_t EndpointFromRequestGet (uint16_t wIndex)

Gets the endpoint status.

Parameters:

wIndex	- Endpoint address and direction

Returns:

A structure with the endpoint status

4.4.6.3.2 SetupDeviceAddressCallback()

void SetupDeviceAddressCallback (void)

Callback function for the address.

Parameters:

None.

Returns:

None.

4.4.6.3.3 SetupDeviceRequestClearFeature()

RETURN_CODE_t SetupDeviceRequestClearFeature (USB_SETUP_REQUEST_t * setupRequestPtr)

Clears the device feature.

Parameters:

*setupRequestPtr	- Pointer to the setup request

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.4 SetupDeviceRequestGetConfiguration()

RETURN_CODE_t SetupDeviceRequestGetConfiguration (void)

Gets the device configuration.

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.5 SetupDeviceRequestGetDescriptor()

RETURN_CODE_t SetupDeviceRequestGetDescriptor (USB_SETUP_REQUEST_t * setupRequestPtr)

Gets the device descriptor.

Parameters:

*setupRequestPtr	- Pointer to the setup request

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.6 SetupDeviceRequestGetStatus()

RETURN_CODE_t SetupDeviceRequestGetStatus (void)

Returns the status of the device features.

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.7 SetupDeviceRequestSetAddress()

RETURN_CODE_t SetupDeviceRequestSetAddress (uint8_t address)

Sets the device address.

Parameters:

address	- Address to be set

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.8 SetupDeviceRequestSetConfiguration()

RETURN_CODE_t SetupDeviceRequestSetConfiguration (uint8_t configurationValue)

Sets the device configuration.

Parameters:

configurationValue	- Configuration value to be set

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.9 SetupDeviceRequestSetFeature()

RETURN_CODE_t SetupDeviceRequestSetFeature (USB_SETUP_REQUEST_t * setupRequestPtr)

Sets the device feature.

Parameters:

*setupRequestPtr	- Pointer to the setup request

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.10 SetupEndpointRequestClearFeature()

RETURN_CODE_t SetupEndpointRequestClearFeature (USB_SETUP_REQUEST_t * setupRequestPtr)

Clears the endpoint feature.

Parameters:

*setupRequestPtr	- Pointer to the setup request

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.11 SetupEndpointRequestGetStatus()

RETURN_CODE_t SetupEndpointRequestGetStatus (USB_SETUP_REQUEST_t * setupRequestPtr)

Gets the endpoint status.

Parameters:

*setupRequestPtr	- Pointer to the setup request

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.12 SetupEndpointRequestSetFeature()

RETURN_CODE_t SetupEndpointRequestSetFeature (USB_SETUP_REQUEST_t * setupRequestPtr)

Sets the endpoint feature.

Parameters:

*setupRequestPtr	- Pointer to the setup request

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.13 SetupEndpointRequestSynchFrame()

RETURN_CODE_t SetupEndpointRequestSynchFrame (void)

Gets the current frame number.

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.14 USB_SetupInterfaceRequestGetDescriptor()

RETURN_CODE_t USB_SetupInterfaceRequestGetDescriptor (USB_SETUP_REQUEST_t * setupRequestPtr)

Setup function for the interface request for class-specific descriptors.

bRequest	wValue	wIndex	wLength	Data
GET_DESCRIPTOR	Type and index	Zero	Length	Descriptor

Parameters:

*setupRequestPtr	- Pointer to the request and its data

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.15 USB_SetupInterfaceRequestGetInterface()

RETURN_CODE_t USB_SetupInterfaceRequestGetInterface (USB_SETUP_REQUEST_t * setupRequestPtr)

Returns the alternate setting for the specified interface.

Format for GET_INTERFACE request according to USB 2.0 specification Ch 9.4.4. Document: Universal Serial Bus Specification for USB 2.0.

bRequest	wValue	wIndex	wLength	Data
GET_INTERFACE	Zero	Interface	One	Alternate setting

Parameters:

*setupRequestPtr	- Pointer to the request and its data

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.16 USB_SetupInterfaceRequestGetStatus()

RETURN_CODE_t USB_SetupInterfaceRequestGetStatus (void)

Returns status for the specified interface.

Get status from interface request according to USB 2.0 specification Ch. 9.4.5.

bRequest	wValue	wIndex	wLength	Data
GET_STATUS	Zero	Interface	Two	Interface

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.17 USB_SetupInterfaceRequestSetInterface()

RETURN_CODE_t USB_SetupInterfaceRequestSetInterface (USB_SETUP_REQUEST_t * setupRequestPtr)

Setup function for the interface request to select the alternate setting.

A request to set interface according to USB 2.0 specification Ch. 9.4.10. Document: Universal Serial Bus Specification for USB 2.0

bRequest	wValue	wIndex	wLength	Data
SET_INTERFACE	Alternate setting	Interface	Zero	None

Parameters:

*setupRequestPtr	- Pointer to the request and its data

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.18 USB_SetupProcessDeviceRequest()

RETURN_CODE_t USB_SetupProcessDeviceRequest (USB_SETUP_REQUEST_t * setupRequestPtr)

Setup function for the device requests.

USB 2.0 Specification Ch 9.4.

bRequest	wValue	wIndex	wLength	Data
CLEAR_FEATURE	Feature selector	Zero	Zero	None
GET_CONFIGURATION	Zero	Zero	One	Config value
GET_DESCRIPTOR	Type and index	Zero or ID	Length	Descriptor
GET_STATUS	Zero	Endpoint	Two	Device status
SET_ADDRESS	Device address	Zero	Zero	None
SET_CONFIGURATION	Config value	Zero	Zero	None
SET_DESCRIPTOR	Type and index	Zero or ID	Length	Descriptor
SET_FEATURE	Feature selector	Zero	Zero	None

Parameters:

setupRequestPtr	- Pointer to the setup request and its data

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.19 USB_SetupProcessEndpointRequest()

RETURN_CODE_t USB_SetupProcessEndpointRequest (USB_SETUP_REQUEST_t * setupRequestPtr)

Setup function for the endpoint requests.

USB 2.0 Specification Ch. 9.4.

bRequest	wValue	wIndex	wLength	Data
CLEAR_FEATURE	Feature selector	Endpoint	Zero	None
GET_STATUS	Zero	Endpoint	Two	Endpoint status
SET_FEATURE	Feature selector	Endpoint	Zero	None
SYNCH_FRAME	Zero	Endpoint	Two	Frame number

Parameters:

*setupRequestPtr	- Pointer to the request and its data

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.6.3.20 USB_SetupProcessInterfaceRequest()

RETURN_CODE_t USB_SetupProcessInterfaceRequest (USB_SETUP_REQUEST_t * setupRequestPtr)

Setup function for the interface requests.

USB 2.0 Specification Ch 9.4.

bRequest	wValue	wIndex	wLength	Data
CLEAR_FEATURE	Feature selector	Interface	Zero	None
GET_INTERFACE	Zero	Interface	One	Alternate interface
GET_STATUS	Zero	Interface	Two	Interface
SET_FEATURE	Feature selector	Interface	Zero	None
SET_INTERFACE	Alternate setting	Interface	Zero	None
GET_DESCRIPTOR	Type and index	Zero	Length	Descriptor

Parameters:

setupRequestPtr	- Pointer to the setup request and its data

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.7 USB Core Transfer

USB core layer implementation file.

4.4.7.1 Module description

USB core layer implementation file.

Version: USB Core Version 1.0.0

4.4.7.1.1 Functions

- RETURN_CODE_t USB_TransferWriteStart (USB_PIPE_t pipe, uint8_t *dataPtr, uint16_t dataSize, bool useZLP, USB_TRANSFER_END_CALLBACK_t callback)
  Sets up the pipe for the write transfers.
- RETURN_CODE_t USB_TransferReadStart (USB_PIPE_t pipe, uint8_t *dataPtr, uint16_t dataSize, bool useZLP, USB_TRANSFER_END_CALLBACK_t callback)
  Sets up the pipe for the read transfers.
- RETURN_CODE_t USB_TransferControlDataSet (uint8_t *dataPtr, uint16_t dataSize, USB_SETUP_ENDOFREQUEST_CALLBACK_t callback)
  Sets up vendor or class control request data transfers.
- RETURN_CODE_t USB_TransferAbort (USB_PIPE_t pipe)
  Aborts an ongoing transfer.
- RETURN_CODE_t USB_TransferHandler (void)
  Handles the different types of packages received or transferred.

4.4.7.2 Function Documentation

4.4.7.2.1 USB_TransferAbort()

RETURN_CODE_t USB_TransferAbort (USB_PIPE_t pipe)

Aborts an ongoing transfer.

Will call the pipe transferEndCallback with the abort status, if configured.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.7.2.2 USB_TransferControlDataSet()

RETURN_CODE_t USB_TransferControlDataSet (uint8_t * dataPtr, uint16_t dataSize, USB_SETUP_ENDOFREQUEST_CALLBACK_t callback)

Sets up vendor or class control request data transfers.

Sets up the pointer and size of the read or write trnasfer in the control data stage.

Parameters:

*dataPtr	- The pointer to the data to read or write
dataSize	- The size of the data to read or write
callback	- Pointer to a function to be called at the end of the control request

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.7.2.3 USB_TransferHandler()

RETURN_CODE_t USB_TransferHandler (void)

Handles the different types of packages received or transferred.

Checks if a setup package is received or if a transaction is completed and which pipe has a completed transaction, then it handles them accordingly. Sends an ACK upon completed transaction confirmation.

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.7.2.4 USB_TransferReadStart()

RETURN_CODE_t USB_TransferReadStart (USB_PIPE_t pipe, uint8_t * dataPtr, uint16_t dataSize, bool useZLP, USB_TRANSFER_END_CALLBACK_t callback)

Sets up the pipe for the read transfers.

Sets up the pipe for the write transfers and checks to see if it is busy. If it is not busy, then the routine resets the pipe, setting it up for read transfer, then starts the transfer.

Parameters:

pipe	- A combination of endpoint address and direction
*dataPtr	- The pointer to the data to read

dataSize	- The size of the data to read
useZLP	- Enable zero-length package at the end of transfer
callback	- A combination of pipe, status and transferred bytes

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.7.2.5 USB_TransferWriteStart()

RETURN_CODE_t USB_TransferWriteStart (USB_PIPE_t pipe, uint8_t * dataPtr, uint16_t dataSize, bool useZLP, USB_TRANSFER_END_CALLBACK_t callback)

Sets up the pipe for the write transfers.

Sets up the pipe for the write transfers and checks to see if it is busy. If it is not busy, then the routine resets the pipe, setting it up for write transfer, then starts the transfer.

Parameters:

pipe	- A combination of endpoint address and direction
*dataPtr	- The pointer to the data to write
dataSize	- The size of the data to write
useZLP	- Enable zero-length package at the end of transfer
callback	- A combination of pipe, status and transferred bytes

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.8 USB Protocol

Common data structures, enumerations and macro definitons based on the USB 2.0 base protocol.

4.4.8.1 Module description

Common data structures, enumerations and macro definitons based on the USB 2.0 base protocol.

Version: USB Protocol Version 1.0.0

4.4.8.1.1 Data structures

- struct USB_PIPE_struct
  Structure for a pipe which has an address and direction.
- struct USB_IAD_DESC_struct
  Standard USB association descriptor structure.
- struct USB_INTERFACE_DESCRIPTOR_struct
  Ch. 9.6.5 Standard USB interface descriptor structure.
- struct USB_STRING_LANG_ID_DESCRIPTOR_struct
  Structure for the USB string Language ID descriptor.
- struct USB_STRING_DESCRIPTOR_struct
  Structure with pointers to the standard USB descriptors.

4.4.8.1.2 Typedefs

- typedef void(* USB_TRANSFER_END_CALLBACK_t) (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)
  Function callback type USB_TRANSFER_END_CALLBACK_t. Callback type used for transfer complete notifications.

- typedef RETURN_CODE_t(* USB_SETUP_PROCESS_CALLBACK_t) (USB_SETUP_REQUEST_t *setupRequestPtr)

  Function callback type USB_SETUP_PROCESS_CALLBACK_t. Callback type used for setup request processing, with a return code to let the stack know to proceed.
- typedef RETURN_CODE_t(* USB_SETUP_STRING_CALLBACK_t) (uint8_t stringIndex, uint16_t langID, uint8_t **descriptorAddressPtr, uint16_t *descriptorLength)

  Function callback type USB_SETUP_STRING_CALLBACK_t. Callback type used for setup request processing a string descriptor, with a return code to let the stack know to proceed.
- typedef void(* USB_SETUP_EVENT_CALLBACK_t) (USB_SETUP_REQUEST_t *setupRequestPtr)

  Function callback type USB_SETUP_EVENT_CALLBACK_t. Callback type used for setup request notifications.
- typedef RETURN_CODE_t(* USB_SETUP_OVERUNDERRUN_CALLBACK_t) (void)

  Function callback type USB_SETUP_OVERUNDERRUN_CALLBACK_t. Callback type used for USB Overrun and Underrun event processing on the control endpoints, with a return code to let the stack know to proceed.
- typedef void(* USB_SETUP_ENDOFREQUEST_CALLBACK_t) (void)

  Function callback type USB_SETUP_ENDOFREQUEST_CALLBACK_t. Callback type used for setup request complete notifications.
- typedef void(* USB_EVENT_CALLBACK_t) (void)

  Function callback type USB_EVENT_CALLBACK_t. Callback type used for USB event notifications.

4.4.8.1.3 Definitions

- #define USB_EP_DIR_IN 0x01u

  Endpoint direction IN.
- #define USB_EP_DIR_OUT 0x00u

  Endpoint direction OUT.
- #define OVERFLOW_EVENT 1u

  Overflow event for the endpoint.
- #define UNDERFLOW_EVENT 2u

  Underflow event for the endpoint.
- #define USB_V2_0 0x0200

  USB Specification version 2.00.
- #define USB_V2_1 0x0201

  USB Specification version 2.01.
- #define NO_SUBCLASS 0x00

  No subclass code.
- #define NO_PROTOCOL 0x00

  No protocol code.
- #define CLASS_IAD 0xEF

  IAD class code.
- #define SUB_CLASS_IAD 0x02

  IAD subclass code.
- #define PROTOCOL_IAD 0x01

  IAD protocol code.
- #define USB_ENDPOINT_FEATURE_HALT 0x00u

USB endpoint feature halt.

- #define DESCRIPTOR_STRING_LENGTH(wstring) (sizeof(wstring) / sizeof(wchar_t) - 1)
  Calculates descriptor length of a UTF-16 string descriptor without the null character.
- #define USB_CONFIG_ATTR_MUST_SET (1u << 7u)
  USB Attribute bitfield for the configuration descriptor.
- #define USB_CONFIG_ATTR_BUS_POWERED (0u << 6u)
  USB Attribute Bus Powered bitfield for the configuration descriptor.
- #define USB_CONFIG_ATTR_SELF_POWERED (1u << 6u)
  USB Attribute Self Powered bitfield for the configuration descriptor.
- #define USB_CONFIG_ATTR_REMOTE_WAKEUP (1u << 5u)
  USB Attribute Remote Wakeup bitfield for the configuration descriptor.
- #define USB_CONFIG_MAX_POWER(ma) (((ma) + 1u) / 2u)
  USB Max Power bitfield for the configuration descriptor.

4.4.8.1.4 Enumerations

- enum USB_ENDPOINT_enum { CONTROL = 0, ISOCHRONOUS = 1, BULK = 2, INTERRUPT = 3, DISABLED = 0xff }
  Defines labels for the different endpoint types as per the USB 2.0 base specification.
- enum USB_TRANSFER_STATUS_enum { USB_PIPE_TRANSFER_OK = 0, USB_PIPE_TRANSFER_BUSY = 1, USB_PIPE_TRANSFER_ABORTED = 2, USB_PIPE_TRANSFER_ERROR = 3 }
  Defines the possible states of a configured transfer.
- enum USB_CONTROL_STATUS_enum { USB_CONTROL_SETUP = 0, USB_CONTROL_DATA_OUT = 1, USB_CONTROL_DATA_IN = 2, USB_CONTROL_ZLP = 3, USB_CONTROL_STALL_REQ = 4 }
  Defines the possible states of a configured control transfer.
- enum USB_REQUEST_DIR_enum { USB_REQUEST_DIR_OUT = 0, USB_REQUEST_DIR_IN = 1 }
  Standard USB enumeration used by setup requests.
- enum USB_REQUEST_TYPE_enum { USB_REQUEST_TYPE_STANDARD = 0, USB_REQUEST_TYPE_CLASS = 1, USB_REQUEST_TYPE_VENDOR = 2 }
  USB request types (bmRequestType).
- enum USB_REQUEST_RECIPIENT_enum { USB_REQUEST_RECIPIENT_DEVICE = 0, USB_REQUEST_RECIPIENT_INTERFACE = 1, USB_REQUEST_RECIPIENT_ENDPOINT = 2, USB_REQUEST_RECIPIENT_OTHER = 3 }
  USB recipient codes (bmRequestType).
- enum USB_REQUEST_ID_enum { USB_REQUEST_GET_STATUS = 0, USB_REQUEST_CLEAR_FEATURE = 1, USB_REQUEST_SET_FEATURE = 3, USB_REQUEST_SET_ADDRESS = 5, USB_REQUEST_GET_DESCRIPTOR = 6, USB_REQUEST_SET_DESCRIPTOR = 7, USB_REQUEST_GET_CONFIGURATION = 8, USB_REQUEST_SET_CONFIGURATION = 9, USB_REQUEST_GET_INTERFACE = 10, USB_REQUEST_SET_INTERFACE = 11, USB_REQUEST_SYNCH_FRAME = 12 }
  Standard USB requests (bRequest).
- enum USB_DESCRIPTOR_TYPE_enum { USB_DESCRIPTOR_TYPE_DEVICE = 1, USB_DESCRIPTOR_TYPE_CONFIGURATION = 2, USB_DESCRIPTOR_TYPE_STRING = 3, USB_DESCRIPTOR_TYPE_INTERFACE = 4, USB_DESCRIPTOR_TYPE_ENDPOINT = 5, USB_DESCRIPTOR_TYPE_DEVICE_QUALIFIER = 6, USB_DESCRIPTOR_TYPE_OTHER_SPEED_CONFIGURATION = 7, USB_DESCRIPTOR_TYPE_INTERFACE_POWER = 8, USB_DESCRIPTOR_TYPE_IAD = 11,

USB_DESCRIPTOR_TYPE_BOS = 15, USB_DESCRIPTOR_TYPE_DEVICE_CAPABILITY = 16,
USB_DESCRIPTOR_TYPE_CLASS = 0x20, USB_DESCRIPTOR_TYPE_VENDOR = 0x40 }

Standard USB descriptor types.

4.4.8.2 Definition Documentation

4.4.8.2.1 CLASS_IAD

#define CLASS_IAD 0xEF

IAD class code.

4.4.8.2.2 DESCRIPTOR_STRING_LENGTH

#define DESCRIPTOR_STRING_LENGTH(wstring) (sizeof(wstring) / sizeof(wchar_t) - 1)

Calculates descriptor length of a UTF-16 string descriptor without the null character.

4.4.8.2.3 NO_PROTOCOL

#define NO_PROTOCOL 0x00

No protocol code.

4.4.8.2.4 NO_SUBCLASS

#define NO_SUBCLASS 0x00

No subclass code.

4.4.8.2.5 OVERFLOW_EVENT

#define OVERFLOW_EVENT 1u

Overflow event for the endpoint.

4.4.8.2.6 PROTOCOL_IAD

#define PROTOCOL_IAD 0x01

IAD protocol code.

4.4.8.2.7 SUB_CLASS_IAD

#define SUB_CLASS_IAD 0x02

IAD subclass code.

4.4.8.2.8 UNDERFLOW_EVENT

#define UNDERFLOW_EVENT 2u

Underflow event for the endpoint.

4.4.8.2.9 USB_CONFIG_ATTR_BUS_POWERED

#define USB_CONFIG_ATTR_BUS_POWERED (0u << 6u)

USB Attribute Bus Powered bitfield for the configuration descriptor.

Bus-Powered

4.4.8.2.10 USB_CONFIG_ATTR_MUST_SET

#define USB_CONFIG_ATTR_MUST_SET (1u << 7u)

USB Attribute bitfield for the configuration descriptor.

Must always be set

4.4.8.2.11 USB_CONFIG_ATTR_REMOTE_WAKEUP

#define USB_CONFIG_ATTR_REMOTE_WAKEUP (1u << 5u)

USB Attribute Remote Wakeup bitfield for the configuration descriptor.

Remote wakeup supported

4.4.8.2.12 USB_CONFIG_ATTR_SELF_POWERED

#define USB_CONFIG_ATTR_SELF_POWERED (1u << 6u)

USB Attribute Self Powered bitfield for the configuration descriptor.

Self-Powered

4.4.8.2.13 USB_CONFIG_MAX_POWER

#define USB_CONFIG_MAX_POWER(ma) (((ma) + 1u) / 2u)

USB Max Power bitfield for the configuration descriptor.

Maximum power in mA

4.4.8.2.14 USB_ENDPOINT_FEATURE_HALT

#define USB_ENDPOINT_FEATURE_HALT 0x00u

USB endpoint feature halt.

4.4.8.2.15 USB_EP_DIR_IN

#define USB_EP_DIR_IN 0x01u

Endpoint direction IN.

4.4.8.2.16 USB_EP_DIR_OUT

#define USB_EP_DIR_OUT 0x00u

Endpoint direction OUT.

4.4.8.2.17 USB_V2_0

#define USB_V2_0 0x0200

USB Specification version 2.00.

4.4.8.2.18 USB_V2_1

#define USB_V2_1 0x0201

USB Specification version 2.01.

4.4.8.3 Typedef Documentation

4.4.8.3.1 USB_EVENT_CALLBACK_t

typedef void(* USB_EVENT_CALLBACK_t) (void)

Function callback type USB_EVENT_CALLBACK_t. Callback type used for USB event notifications.

Parameters:

None.

Returns:

None.

4.4.8.3.2 USB_SETUP_ENDOFREQUEST_CALLBACK_t

typedef void(* USB_SETUP_ENDOFREQUEST_CALLBACK_t) (void)

Function callback type USB_SETUP_ENDOFREQUEST_CALLBACK_t. Callback type used for setup request complete notifications.

Parameters:

None.

Returns:

None.

4.4.8.3.3 USB_SETUP_EVENT_CALLBACK_t

typedef void(* USB_SETUP_EVENT_CALLBACK_t) (USB_SETUP_REQUEST_t *setupRequestPtr)

Function callback type USB_SETUP_EVENT_CALLBACK_t. Callback type used for setup request notifications.

Parameters:

*setupRequestPtr	- Pointer to the current setup request data structure

Returns:

None.

4.4.8.3.4 USB_SETUP_OVERUNDERRUN_CALLBACK_t

typedef RETURN_CODE_t(* USB_SETUP_OVERUNDERRUN_CALLBACK_t) (void)

Function callback type USB_SETUP_OVERUNDERRUN_CALLBACK_t. Callback type used for USB Overrun and Underrun event processing on the control endpoints, with a return code to let the stack know to proceed.

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.8.3.5 USB_SETUP_PROCESS_CALLBACK_t

typedef RETURN_CODE_t(* USB_SETUP_PROCESS_CALLBACK_t) (USB_SETUP_REQUEST_t *setupRequestPtr)

Function callback type USB_SETUP_PROCESS_CALLBACK_t. Callback type used for setup request processing, with a return code to let the stack know to proceed.

Parameters:

*setupRequestPtr	- Pointer to the current setup request data structure

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.8.3.6 USB_SETUP_STRING_CALLBACK_t

typedef RETURN_CODE_t(* USB_SETUP_STRING_CALLBACK_t) (uint8_t stringIndex, uint16_t langID, uint8_t **descriptorAddressPtr, uint16_t *descriptorLength)

Function callback type USB_SETUP_STRING_CALLBACK_t. Callback type used for setup request processing a string descriptor, with a return code to let the stack know to proceed.

Parameters:

stringIndex	- Specifies which string of device information is requested
langID	- Which language the string requested must be in
**descriptorAddressPtr	- Pointer to write string descriptor address to
*descriptorLength	- pointer to the size of the descriptor

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.8.3.7 USB_TRANSFER_END_CALLBACK_t

typedef void(* USB_TRANSFER_END_CALLBACK_t) (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)

Function callback type USB_TRANSFER_END_CALLBACK_t. Callback type used for transfer complete notifications.

Parameters:

pipe	- Address and direction of the pipe used for the transfer
status	- Status of the completed transfer, USB_PIPE_TRANSFER_OK when successful
bytesTransferred	- Number of bytes transferred

Returns:

None.

4.4.8.4 Enumeration Type Documentation

4.4.8.4.1 USB_CONTROL_STATUS_enum

enum USB_CONTROL_STATUS_enum

Defines the possible states of a configured control transfer.

USB_CONTROL_SETUP	Wait a SETUP packet
USB_CONTROL_DATA_OUT	Wait an OUT data packet
USB_CONTROL_DATA_IN	Wait an IN data packet
USB_CONTROL_ZLP	Wait an IN or OUT ZLP packet
USB_CONTROL_STALL_REQ	STALL enabled on IN and OUT packets

4.4.8.4.2 USB_DESCRIPTOR_TYPE_enum

enum USB_DESCRIPTOR_TYPE_enum

Standard USB descriptor types.

USB_DESCRIPTOR_TYPE_DEVICE	Provides essential information about the USB device
USB_DESCRIPTOR_TYPE_CONFIGURATION	Describes a specific configuration of the device
USB_DESCRIPTOR_TYPE_STRING	Contains human-readable string information
USB_DESCRIPTOR_TYPE_INTERFACE	Specifies the characteristics of a single interface within a configuration
USB_DESCRIPTOR_TYPE_ENDPOINT	Defines the characteristics of a specific endpoint within an interface
USB_DESCRIPTOR_TYPE_DEVICE_QUALIFIER	For high-speed USB devices, informs the host if the device has different device information for when operating in full-speed
USB_DESCRIPTOR_TYPE_OTHER_SPEED_CONFIGURATION	Provides details about the configuration of a device for a specific speed when operating at high-speed
USB_DESCRIPTOR_TYPE_INTERFACE_POWER	Specifies the amount of power that an interface of a USB device consumes when it's active
USB_DESCRIPTOR_TYPE_IAD	Interface Association Descriptor which associates a group of interfaces that form a single function, helping to manage composite devices

USB_DESCRIPTOR_TYPE_BOS	Binary Object Store Descriptor which provides information about the capabilities of the device
USB_DESCRIPTOR_TYPE_DEVICE_CAPABILITY	Describes specific capabilities and features supported by the device, such as USB Power Delivery or USB 2.0 features
USB_DESCRIPTOR_TYPE_CLASS	Class descriptor types are from 0x20 to 0x3f
USB_DESCRIPTOR_TYPE_VENDOR	Vendor descriptor types are from 0x40 to 0x5f

4.4.8.4.3 USB_ENDPOINT_enum

enum USB_ENDPOINT_enum

Defines labels for the different endpoint types as per the USB 2.0 base specification.

CONTROL	Control transfer type
ISOCHRONOUS	Isochronous transfer type
BULK	Bulk transfer type
INTERRUPT	Interrupt transfer type
DISABLED	Endpoint disabled

4.4.8.4.4 USB_REQUEST_DIR_enum

enum USB_REQUEST_DIR_enum

Standard USB enumeration used by setup requests.

USB_REQUEST_DIR_OUT	Setup request has direction OUT
USB_REQUEST_DIR_IN	Setup request has direction IN

4.4.8.4.5 USB_REQUEST_ID_enum

enum USB_REQUEST_ID_enum

Standard USB requests (bRequest).

USB_REQUEST_GET_STATUS	Retrieve status information, such as device power status or endpoint stall conditions
USB_REQUEST_CLEAR_FEATURE	Clear or reset specific device or component features, like clearing an endpoint halt condition
USB_REQUEST_SET_FEATURE	Set or enable specific device or component features, often used to configure device behavior
USB_REQUEST_SET_ADDRESS	Assign a unique device address during enumeration
USB_REQUEST_GET_DESCRIPTOR	Request descriptor information, specifying the type of descriptor being requested
USB_REQUEST_SET_DESCRIPTOR	Update certain descriptors
USB_REQUEST_GET_CONFIGURATION	Retrieve the currently selected device configuration
USB_REQUEST_SET_CONFIGURATION	Select a specific device configuration
USB_REQUEST_GET_INTERFACE	Retrieve the currently selected alternate setting of an interface
USB_REQUEST_SET_INTERFACE	Select a specific alternate setting for an interface
USB_REQUEST_SYNCH_FRAME	Retrieve synchronization information for isochronous transfers

4.4.8.4.6 USB_REQUEST_RECIPIENT_enum

enum USB_REQUEST_RECIPIENT_enum

USB recipient codes (bmRequestType).

USB_REQUEST_RECIPIENT_DEVICE	Request is directed at the entire USB device
USB_REQUEST_RECIPIENT_INTERFACE	Request is directed at an interface within the USB device

USB_REQUEST_RECIPIENT_ENDPOINT	Request is directed at an endpoint within an interface
USB_REQUEST_RECIPIENT_OTHER	Request is directed at another specific recipient

4.4.8.4.7 USB_REQUEST_TYPE_enum

enum USB_REQUEST_TYPE_enum

USB request types (bmRequestType).

USB_REQUEST_TYPE_STANDARD	Standard USB request defined in the USB specification
USB_REQUEST_TYPE_CLASS	Class-specific request defined in the USB Class Specification
USB_REQUEST_TYPE_VENDOR	Vendor-specific request not defined in the USB Specification

4.4.8.4.8 USB_TRANSFER_STATUS_enum

enum USB_TRANSFER_STATUS_enum

Defines the possible states of a configured transfer.

READ WRITE LAYER

USB_PIPE_TRANSFER_OK	Successful transfer on the pipe
USB_PIPE_TRANSFER_BUSY	The pipe is busy
USB_PIPE_TRANSFER_ABORTED	Transfer aborted on the pipe
USB_PIPE_TRANSFER_ERROR	Failure during transfer on the pipe

4.4.9 USB Human Interface Device (HID)

Contains the prototypes and data types for the generic HID application drivers.

4.4.9.1 Module description

Contains the prototypes and data types for the generic HID application drivers.

Version: USB Device Stack HID Driver Version 1.0.0

4.4.9.1.1 Data structures

- struct USB_HID_DESCRIPTOR_t
  Type defines for a standard HID descriptor.
- struct USB_HID_REPORT_DESCRIPTOR_t
  Report descriptor for a HID application.
- struct USB_MOUSE_REPORT_DATA_t
  Type defines for a standard mouse input report.
- struct USB_KEYBOARD_REPORT_DATA_t
  Type defines for a standard keyboard input report.

4.4.9.1.2 Typedefs

- typedef void(* USB_HID_REPORT_CALLBACK_t) (uint16_t report)
  Defines a type for registering a callback for the HID report.

4.4.9.1.3 Functions

- void USB_HIDReportUpdatedCallbackRegister (USB_HID_REPORT_CALLBACK_t callback)
  Registers a callback to the HID report.
- void USB_HIDReportUpdatedCallback (void)
  Checks if a callback is registered and calls the End Of Request function.
- void USB_HIDInitialize (uint8_t *ratePtr, uint8_t *protocolPtr, USB_HID_REPORT_DESCRIPTOR_t *reportPtr)

Registers the rate, protocol and report descriptor for HID.

- RETURN_CODE_t USB_HIDRequestHandler (USB_SETUP_REQUEST_t *setupRequestPtr)
  Initializes the HID class and performs control transfers.

4.4.9.2 Typedef Documentation

4.4.9.2.1 USB_HID_REPORT_CALLBACK_t

typedef void(* USB_HID_REPORT_CALLBACK_t) (uint16_t report)

Defines a type for registering a callback for the HID report.

4.4.9.3 Function Documentation

4.4.9.3.1 USB_HIDInitialize()

void USB_HIDInitialize (uint8_t * ratePtr, uint8_t * protocolPtr, USB_HID_REPORT_DESCRIPTOR_t * reportPtr)

Registers the rate, protocol and report descriptor for HID.

Parameters:

ratePtr	- Pointer to rate
protocolPtr	- Pointer to protocol
reportPtr	- Pointer to report descriptor

Returns:

None.

4.4.9.3.2 USB_HIDReportUpdatedCallback()

void USB_HIDReportUpdatedCallback (void)

Checks if a callback is registered and calls the End Of Request function.

Parameters:

None.

Returns:

None.

4.4.9.3.3 USB_HIDReportUpdatedCallbackRegister()

void USB_HIDReportUpdatedCallbackRegister (USB_HID_REPORT_CALLBACK_t callback)

Registers a callback to the HID report.

Parameters:

callback	- Callback to the report updated function

Returns:

None.

4.4.9.3.4 USB_HIDRequestHandler()

RETURN_CODE_t USB_HIDRequestHandler (USB_SETUP_REQUEST_t * setupRequestPtr)

Initializes the HID class and performs control transfers.

Parameters:

setupRequestPtr	- Pointer to the Setup Request struct

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.10 USB Human Interface Device (HID) Keyboard

Contains the prototypes and data types for the USB Keyboard application drivers.

4.4.10.1 Module description

Contains the prototypes and data types for the USB Keyboard application drivers.

Version: USB Device Stack HID Driver Version 1.0.0

4.4.10.1.1 Keyboard input report sizes

Macros for the input report for a standard HID keyboard.

- void USB_HIDKeyboardInitialize (USB_HID_REPORT_DESCRIPTOR_t *reportPtr, USB_HID_REPORT_CALLBACK_t callback)
  Sets up the keyboard application for use with the HID class.
- RETURN_CODE_t USB_HIDKeyModifierDown (uint8_t modifierID)
  Updates the keyboard report by adding the pressed modifier key in the modifier byte in the report array.
- RETURN_CODE_t USB_HIDKeyModifierUp (uint8_t modifierID)
  Updates the keyboard report by removing the released modifier key in the modifier byte in the report array.
- int8_t USB_HIDKeyCodeIndexGet (uint8_t keyID)
  Checks the report array to see that the key is already present.
- RETURN_CODE_t USB_HIDKeyPressDown (uint8_t keyID)
  Updates the keyboard report by adding the pressed key at the first available byte in the report array.
- RETURN_CODE_t USB_HIDKeyPressUp (uint8_t keyID)
  Updates the keyboard report by removing the released key and shifts the registered keys towards the beginning of the report array.
- #define USB_HID_KEYBOARD_REPORT_KEYNUM 6u
- #define USB_HID_KEYBOARD_REPORT_SIZE (USB_HID_KEYBOARD_REPORT_KEYNUM + 2u)

4.4.10.2 Definition Documentation

4.4.10.2.1 USB_HID_KEYBOARD_REPORT_KEYNUM

#define USB_HID_KEYBOARD_REPORT_KEYNUM 6u

4.4.10.2.2 USB_HID_KEYBOARD_REPORT_SIZE

#define USB_HID_KEYBOARD_REPORT_SIZE (USB_HID_KEYBOARD_REPORT_KEYNUM + 2u)

4.4.10.3 Function Documentation

4.4.10.3.1 USB_HIDKeyboardInitialize()

void USB_HIDKeyboardInitialize (USB_HID_REPORT_DESCRIPTOR_t * reportPtr, USB_HID_REPORT_CALLBACK_t callback)

Sets up the keyboard application for use with the HID class.

Parameters:

reportPtr	- Pointer to report descriptor
callback	- Callback for registering the Set Report function

Returns:

None.

4.4.10.3.2 USB_HIDKeyCodeIndexGet()

int8_t USB_HIDKeyCodeIndexGet (uint8_t keyID)

Checks the report array to see that the key is already present.

Parameters:

keyID	- ID of the key being pressed

Returns:

HID_KEYID_NOT_FOUND or location in array where the key is already present

4.4.10.3.3 USB_HIDKeyModifierDown()

RETURN_CODE_t USB_HIDKeyModifierDown (uint8_t modifierID)

Updates the keyboard report by adding the pressed modifier key in the modifier byte in the report array.

Parameters:

modifierID	- ID of modifier key being pressed

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.10.3.4 USB_HIDKeyModifierUp()

RETURN_CODE_t USB_HIDKeyModifierUp (uint8_t modifierID)

Updates the keyboard report by removing the released modifier key in the modifier byte in the report array.

Parameters:

modifierID	- ID of modifier key being released

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.10.3.5 USB_HIDKeyPressDown()

RETURN_CODE_t USB_HIDKeyPressDown (uint8_t keyID)

Updates the keyboard report by adding the pressed key at the first available byte in the report array.

Parameters:

keyID	- ID of key being pressed

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.10.3.6 USB_HIDKeyPressUp()

RETURN_CODE_t USB_HIDKeyPressUp (uint8_t keyID)

Updates the keyboard report by removing the released key and shifts the registered keys towards the beginning of the report array.

Parameters:

keyID	- ID of key being released

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.11 USB Human Interface Device (HID) Mouse

Contains the prototypes and data types for the USB Mouse application drivers.

4.4.11.1 Module description

Contains the prototypes and data types for the USB Mouse application drivers.

Version: USB Device Stack Driver Version 1.0.0

4.4.11.1.1 Definitions

- #define USB_HID_MOUSE_REPORT_SIZE 8
  Size of the report for a standard HID mouse.

4.4.11.1.2 Functions

- void USB_HIDMouseInitialize (USB_HID_REPORT_DESCRIPTOR_t *reportPtr)
  Sets up the mouse application for use with the HID class.
- RETURN_CODE_t USB_HIDMouseMove (int8_t x_position, int8_t y_position)
  Registers mouse movement and sends its coordinates to the host.
- RETURN_CODE_t USB_HIDMouseButton (bool buttonState, uint8_t button)
  Registers the button and its state and sends it to the host.
- RETURN_CODE_t USB_HIDMouseButtonLeft (bool buttonState)
  Registers the button state of the left button.
- RETURN_CODE_t USB_HIDMouseButtonRight (bool buttonState)
  Registers the button state of the right button.
- RETURN_CODE_t USB_HIDMouseButtonMiddle (bool buttonState)
  Registers the button state of the middle button.

4.4.11.1.3 HID Mouse button state

Macros to signal the button state.

- #define HID_MOUSE_BUTTON_DOWN true
- #define HID_MOUSE_BUTTON_UP false

4.4.11.1.4 HID Mouse buttons

Macros for the mouse buttons.

- #define HID_MOUSE_LEFT_BUTTON 0x01u
- #define HID_MOUSE_RIGHT_BUTTON 0x02u
- #define HID_MOUSE_MIDDLE_BUTTON 0x04u

4.4.11.2 Definition Documentation

4.4.11.2.1 HID_MOUSE_BUTTON_DOWN

#define HID_MOUSE_BUTTON_DOWN true

4.4.11.2.2 HID_MOUSE_BUTTON_UP

#define HID_MOUSE_BUTTON_UP false

4.4.11.2.3 HID_MOUSE_LEFT_BUTTON

#define HID_MOUSE_LEFT_BUTTON 0x01u

4.4.11.2.4 HID_MOUSE_MIDDLE_BUTTON

#define HID_MOUSE_MIDDLE_BUTTON 0x04u

4.4.11.2.5 HID_MOUSE_RIGHT_BUTTON

#define HID_MOUSE_RIGHT_BUTTON 0x02u

4.4.11.2.6 USB_HID_MOUSE_REPORT_SIZE

#define USB_HID_MOUSE_REPORT_SIZE 8

Size of the report for a standard HID mouse.

4.4.11.3 Function Documentation

4.4.11.3.1 USB_HIDMouseButton()

RETURN_CODE_t USB_HIDMouseButton (bool buttonState, uint8_t button)

Registers the button and its state and sends it to the host.

Parameters:

buttonState	- Boolean value indicating if the button is pressed or not
button	- Parameter for which button is pressed

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.11.3.2 USB_HIDMouseButtonLeft()

RETURN_CODE_t USB_HIDMouseButtonLeft (bool buttonState)

Registers the button state of the left button.

Parameters:

buttonState	- Boolean value indicating if the button is pressed or not

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.11.3.3 USB_HIDMouseButtonMiddle()

RETURN_CODE_t USB_HIDMouseButtonMiddle (bool buttonState)

Registers the button state of the middle button.

Parameters:

buttonState	- Boolean value indicating if the button is pressed or not

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.11.3.4 USB_HIDMouseButtonRight()

RETURN_CODE_t USB_HIDMouseButtonRight (bool buttonState)

Registers the button state of the right button.

Parameters:

buttonState	- Boolean value indicating if the button is pressed or not

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.11.3.5 USB_HIDMouseInitialize()

void USB_HIDMouseInitialize (USB_HID_REPORT_DESCRIPTOR_t * reportPtr)

Sets up the mouse application for use with the HID class.

Parameters:

reportPtr	- Pointer to report descriptor

Returns:

None.

4.4.11.3.6 USB_HIDMouseMove()

RETURN_CODE_t USB_HIDMouseMove (int8_t x_position, int8_t y_position)

Registers mouse movement and sends its coordinates to the host.

Parameters:

x_position	- Relative position in X direction
y_position	- Relative position in Y direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.12 USB Human Interface Device (HID) Transfer

Contains the prototypes and data types for the USB HID transfer application drivers.

4.4.12.1 Module description

Contains the prototypes and data types for the USB HID transfer application drivers.

Version: USB Device Stack HID Driver Version 1.0.0

4.4.12.1.1 Functions

- RETURN_CODE_t USB_HIDKeyboardReportInSend (USB_KEYBOARD_REPORT_DATA_t *data)
  Sends a HID keyboard input report to the interrupt IN endpoint.
- void USB_HIDKeyboardInputReportSentCallback (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)
  Verifies if a transfer was completed.
- RETURN_CODE_t USB_HIDMouseReportInSend (USB_MOUSE_REPORT_DATA_t *data)
  Sends a HID mouse input report to the interrupt IN endpoint.

4.4.12.2 Function Documentation

4.4.12.2.1 USB_HIDKeyboardInputReportSentCallback()

void USB_HIDKeyboardInputReportSentCallback (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)

Verifies if a transfer was completed.

Parameters:

pipe	- The pipe that data is being sent on
status	- Status of the transfer

bytesTransferred	- Number of bytes transferred

Returns:

None.

4.4.12.2.2 USB_HIDKeyboardReportInSend()

RETURN_CODE_t USB_HIDKeyboardReportInSend (USB_KEYBOARD_REPORT_DATA_t * data)

Sends a HID keyboard input report to the interrupt IN endpoint.

Parameters:

data	- Keyboard input report data

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.12.2.3 USB_HIDMouseReportInSend()

RETURN_CODE_t USB_HIDMouseReportInSend (USB_MOUSE_REPORT_DATA_t * data)

Sends a HID mouse input report to the interrupt IN endpoint.

Parameters:

data	- Mouse input report data

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.13 USB Peripheral Hardware Abstraction Layer (HAL)

Interface for a USB peripheral module that needs to be implemented by a device-specific USB module driver.

4.4.13.1 Module description

Interface for a USB peripheral module that needs to be implemented by a device-specific USB module driver.

Version: USB Device Stack HAL Driver Version 1.0.0

4.4.13.1.1 Modules

- USB Peripheral AVR DU

  This file encompasses all the register settings of the AVR DU device in the form of inline functions. It also abstracts the Read-Modify-Write loop for STATUS registers, which is required, as the hardware and software can both write into the STATUS register.

- USB Peripheral Endpoint

  API module for usb_peripheral_endpoint covering endpoint-related functions.

- USB Peripheral Read/Write

  API module for usb_peripheral covering low-level USB transaction functions.

4.4.13.1.2 Data structures

- struct USB_CONTROL_TRANSFER_t

  The data structure for internally handling control transfers, either IN or OUT.

4.4.13.1.3 Functions

- bool USB_SetupIsReceived (void)

  Detects if the Setup event was received.

- bool USB_EventSOFIsReceived (void)
  Detects if the Start-of-Frame (SOF) event was received.
- void USB_EventSOFClear (void)
  Clears the SOF event.
- bool USB_EventResetIsReceived (void)
  Detects if the Reset event was received.
- void USB_EventResetClear (void)
  Clears the Reset event.
- uint8_t USB_EventOverUnderflowIsReceived (void)
  Detects if an Overflow and/or Underflow event was received.
- uint8_t USB_ControlOverUnderflowIsReceived (void)
  Detects if an Overflow and/or Underflow event was received on the control endpoints.
- void USB_EventOverUnderflowClear (void)
  Clears the Over/Underflow event.
- bool USB_EventSuspendIsReceived (void)
  Detects if a Suspend event was received.
- void USB_EventSuspendClear (void)
  Clears the Suspend event.
- bool USB_EventResumeIsReceived (void)
  Detects if a Resume event was received.
- void USB_EventResumeClear (void)
  Clears the Resume event.
- bool USB_EventStalledIsReceived (void)
  Detects if a Stalled event was received.
- void USB_EventStalledClear (void)
  Clears the Stalled event.
- void USB_BusAttach (void)
  Attaches the device to the USB bus.
- void USB_BusDetach (void)
  Detaches the device from the USB bus.
- bool USB_IsBusAttached (void)
  Checks if the device is attached to the USB bus not.
- void USB_DeviceAddressConfigure (uint8_t deviceAddress)
  Sets the device address.
- uint16_t USB_FrameNumberGet (void)
  Gets the current frame number.
- RETURN_CODE_t USB_ControlEndpointsInit (void)
  Ensures correct control endpoint initialization.
- RETURN_CODE_t USB_ControlSetupReceived (void)
  Verifies the received control setup.
- RETURN_CODE_t USB_ControlTransactionComplete (USB_PIPE_t pipe)
  Handles completed transactions on the control endpoints. Checks and verifies data OUT, data IN, ZLP OUT and ZLP IN.

- RETURN_CODE_t USB_ControlTransferZLP (uint8_t direction)
  Sends ZLP OUT and ZLP IN transactions on the control endpoints.
- RETURN_CODE_t USB_ControlTransferReset (void)
  Ensures correct control transfer reset.
- RETURN_CODE_t USB_ControlTransferDataSet (uint8_t *dataPtr, uint16_t dataSize)
  Updates the transfer data pointer and size in ControlTransfer.
- RETURN_CODE_t USB_ControlTransferDataWriteBuffer (uint8_t *dataPtr, uint8_t dataSize)
  Copies data to the transfer buffer and sets the transfer data pointer and size in ControlTransfer.
- void USB_ControlEndOfRequestCallbackRegister (USB_SETUP_ENDOFREQUEST_CALLBACK_t callback)
  Sets the callback for end of a control request.
- void USB_ControlProcessSetupCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)
  Sets the callback for the setup processing.
- void USB_ControlOverUnderRunCallbackRegister (USB_SETUP_OVERUNDERRUN_CALLBACK_t callback)
  Sets the callback for a control overrun or underrun.
- RETURN_CODE_t USB_ControlProcessOverUnderflow (uint8_t overunderflow)
  Handles the control Over/Underflow events.
- RETURN_CODE_t USB_HandleEventStalled (USB_PIPE_t pipe)
  Handles the Stall events.
- void USB_PeripheralInitialize (void)
  Enables the peripheral and the frame number, enables and resets FIFO, sets the endpoint table address and max endpoints.
- void USB_PeripheralDisable (void)
  Disables the USB peripheral and aborts any ongoing transaction.
- static ALWAYS_INLINE uint16_t USB_FrameNumGet (void)
  Gets the current frame number.

4.4.13.2 Function Documentation

4.4.13.2.1 USB_BusAttach()

void USB_BusAttach (void)

Attaches the device to the USB bus.

Parameters:

None.

Returns:

None.

4.4.13.2.2 USB_BusDetach()

void USB_BusDetach (void)

Detaches the device from the USB bus.

Parameters:

None.

Returns:

None.

4.4.13.2.3 USB_ControlEndOfRequestCallbackRegister()

void USB_ControlEndOfRequestCallbackRegister (USB_SETUP_ENDOFREQUEST_CALLBACK_t callback)

Sets the callback for end of a control request.

Parameters:

callback	- The function to call for the end of a control request

Returns:

None.

4.4.13.2.4 USB_ControlEndpointsInit()

RETURN_CODE_t USB_ControlEndpointsInit (void)

Ensures correct control endpoint initialization.

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.13.2.5 USB_ControlOverUnderflowIsReceived()

uint8_t USB_ControlOverUnderflowIsReceived (void)

Detects if an Overflow and/or Underflow event was received on the control endpoints.

Parameters:

None.

Returns:

A value representing the events received

4.4.13.2.6 USB_ControlOverUnderRunCallbackRegister()

void USB_ControlOverUnderRunCallbackRegister (USB_SETUP_OVERUNDERRUN_CALLBACK_t callback)

Sets the callback for a control overrun or underrun.

Parameters:

callback	- The function to call on a control overrun or underrun

Returns:

None.

4.4.13.2.7 USB_ControlProcessOverUnderflow()

RETURN_CODE_t USB_ControlProcessOverUnderflow (uint8_t overunderflow)

Handles the control Over/Underflow events.

Parameters:

overunderflow	- A value representing overflow or underflow

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.13.2.8 USB_ControlProcessSetupCallbackRegister()

void USB_ControlProcessSetupCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)

Sets the callback for the setup processing.

Parameters:

callback	- The function to call for the setup processing

Returns:

None.

4.4.13.2.9 USB_ControlSetupReceived()

RETURN_CODE_t USB_ControlSetupReceived (void)

Verifies the received control setup.

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.13.2.10 USB_ControlTransactionComplete()

RETURN_CODE_t USB_ControlTransactionComplete (USB_PIPE_t pipe)

Handles completed transactions on the control endpoints. Checks and verifies data OUT, data IN, ZLP OUT and ZLP IN.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.13.2.11 USB_ControlTransferDataSet()

RETURN_CODE_t USB_ControlTransferDataSet (uint8_t * dataPtr, uint16_t dataSize)

Updates the transfer data pointer and size in ControlTransfer.

Parameters:

*dataPtr	- Pointer to new data
dataSize	- Number of elements in the array

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.13.2.12 USB_ControlTransferDataWriteBuffer()

RETURN_CODE_t USB_ControlTransferDataWriteBuffer (uint8_t * dataPtr, uint8_t dataSize)

Copies data to the transfer buffer and sets the transfer data pointer and size in ControlTransfer.

Parameters:

*dataPtr	- Pointer to data to copy
dataSize	- Number of elements in the array

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.13.2.13 USB_ControlTransferReset()

RETURN_CODE_t USB_ControlTransferReset (void)

Ensures correct control transfer reset.

Parameters:

None.

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.13.2.14 USB_ControlTransferZLP()

RETURN_CODE_t USB_ControlTransferZLP (uint8_t direction)

Sends ZLP OUT and ZLP IN transactions on the control endpoints.

Parameters:

direction	- The endpoint direction to send the ZLP

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.13.2.15 USB_DeviceAddressConfigure()

void USB_DeviceAddressConfigure (uint8_t deviceAddress)

Sets the device address.

Parameters:

deviceAddress	- Device address to set

Returns:

None.

4.4.13.2.16 USB_EventOverUnderflowClear()

void USB_EventOverUnderflowClear (void)

Clears the Over/Underflow event.

Parameters:

None.

Returns:

None.

4.4.13.2.17 USB_EventOverUnderflowIsReceived()

uint8_t USB_EventOverUnderflowIsReceived (void)

Detects if an Overflow and/or Underflow event was received.

Parameters:

None.

Returns:

A value representing the events received

4.4.13.2.18 USB_EventResetClear()

void USB_EventResetClear (void)

Clears the Reset event.

Parameters:

None.

Returns:

None.

4.4.13.2.19 USB_EventResetIsReceived()

bool USB_EventResetIsReceived (void)

Detects if the Reset event was received.

Parameters:

None.

Return values:

1	- Reset event was received
0	- Reset event was not received

4.4.13.2.20 USB_EventResumeClear()

void USB_EventResumeClear (void)

Clears the Resume event.

Parameters:

None.

Returns:

None.

4.4.13.2.21 USB_EventResumeIsReceived()

bool USB_EventResumeIsReceived (void)

Detects if a Resume event was received.

Parameters:

None.

Return values:

0	- Resume event was not received
1	- Resume event was received

4.4.13.2.22 USB_EventSOFClear()

void USB_EventSOFClear (void)

Clears the SOF event.

Parameters:

None.

Returns:

None.

4.4.13.2.23 USB_EventSOFIsReceived()

bool USB_EventSOFIsReceived (void)

Detects if the Start-of-Frame (SOF) event was received.

Parameters:

None.

Return values:

0	- SOF event was not received
1	- SOF event was received

4.4.13.2.24 USB_EventStalledClear()

void USB_EventStalledClear (void)

Clears the Stalled event.

Parameters:

None.

Returns:

None.

4.4.13.2.25 USB_EventStalledIsReceived()

bool USB_EventStalledIsReceived (void)

Detects if a Stalled event was received.

Parameters:

None.

Return values:

0	- Stalled event was not received
1	- Stalled event was received

4.4.13.2.26 USB_EventSuspendClear()

void USB_EventSuspendClear (void)

Clears the Suspend event.

Parameters:

None.

Returns:

None.

4.4.13.2.27 USB_EventSuspendIsReceived()

bool USB_EventSuspendIsReceived (void)

Detects if a Suspend event was received.

Parameters:

None.

Returns:

A boolean value representing the Suspend event received condition

Return values:

0	- Suspend event was not received
1	- Suspend event was received

4.4.13.2.28 USB_FrameNumberGet()

uint16_t USB_FrameNumberGet (void)

Gets the current frame number.

Parameters:

None.

Returns:

15-bit frame number

4.4.13.2.29 USB_FrameNumGet()

static ALWAYS_INLINE uint16_t USB_FrameNumGet (void)[static]

Gets the current frame number.

Parameters:

None.

Returns:

15-bit frame number

4.4.13.2.30 USB_HandleEventStalled()

RETURN_CODE_t USB_HandleEventStalled (USB_PIPE_t pipe)

Handles the Stall events.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.13.2.31 USB_IsBusAttached()

bool USB_IsBusAttached (void)

Checks if the device is attached to the USB bus not.

Parameters:

None.

Return values:

0	- USB bus is not attached
1	- USB bus is attached

4.4.13.2.32 USB_PeripheralDisable()

void USB_PeripheralDisable (void)

Disables the USB peripheral and aborts any ongoing transaction.

Parameters:

None.

Returns:

None.

4.4.13.2.33 USB_PeripheralInitialize()

void USB_PeripheralInitialize (void)

Enables the peripheral and the frame number, enables and resets FIFO, sets the endpoint table address and max endpoints.

Parameters:

None.

Returns:

None.

4.4.13.2.34 USB_SetupIsReceived()

bool USB_SetupIsReceived (void)

Detects if the Setup event was received.

Parameters:

None.

Return values:

0	- Setup event was not received
1	- Setup event was received

4.4.14 USB Peripheral AVR DU

This file encompasses all the register settings of the AVR DU device in the form of inline functions. It also abstracts the Read-Modify-Write loop for STATUS registers, which is required, as the hardware and software can both write into the STATUS register.

4.4.14.1 Module description

This file encompasses all the register settings of the AVR DU device in the form of inline functions. It also abstracts the Read-Modify-Write loop for STATUS registers, which is required, as the hardware and software can both write into the STATUS register.

Version: USB Device Stack HAL Driver Version 1.0.0

4.4.14.1.1 Data structures

- struct USB_ENDPOINT_TABLE_struct

  Represents the endpoint configuration table based on the number of endpoints in use. The table data structure is defined by USB_EP_TABLE_struct in the device header file, modified to support configuration of size from USB_EP_NUM.

- struct USB_PIPE_TRANSFER_struct

  Represents a transfer created for a pipe, either IN or OUT.

4.4.14.1.2 Definitions

- #define ALWAYS_INLINE __attribute__((always_inline)) inline
  Alias that makes always inline function definitions more readable.

4.4.14.1.3 Functions

- static ALWAYS_INLINE void WaitUntilRMWDone (void)
  Waits until a Read-Modify-Write operation is done. This blocking wait operation is expected to complete within 14 clock cycles.
- static ALWAYS_INLINE void USB_EndPointOutDisable (uint8_t endpointAddress)
  Disables the OUT endpoint with the given address.
- static ALWAYS_INLINE void USB_EndPointInDisable (uint8_t endpointAddress)
  Disables the IN endpoint with the given address.
- static ALWAYS_INLINE bool USB_EndPointOutIsEnabled (uint8_t endpointAddress)
  Checks if the OUT endpoint at the given address is enabled.
- static ALWAYS_INLINE bool USB_EndPointInIsEnabled (uint8_t endpointAddress)
  Checks if the IN endpoint at the given address is enabled.
- static ALWAYS_INLINE uint8_t USB_EndPointOutTypeConfigGet (uint8_t endpointAddress)
  Gets the OUT endpoint configuration at the given address.
- static ALWAYS_INLINE uint8_t USB_EndPointInTypeConfigGet (uint8_t endpointAddress)
  Gets the IN endpoint configuration at the given address.
- static ALWAYS_INLINE void USB_EndpointOutControlSet (uint8_t endpointAddress, uint8_t value)
  Sets endpoint control OUT.
- static ALWAYS_INLINE void USB_EndpointInControlSet (uint8_t endpointAddress, uint8_t value)
  Sets endpoint control IN.
- static ALWAYS_INLINE void USB_EndpointOutStatusClear (uint8_t endpointAddress)
  Clears OUT endpoint status.
- static ALWAYS_INLINE void USB_EndpointInStatusClear (uint8_t endpointAddress)
  Clears IN endpoint status.
- static ALWAYS_INLINE void USB_EndpointOutDefaultSizeSet (uint8_t endpointAddress, uint8_t endpointSizeConfig)
  Sets the endpoint size for a default type OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInDefaultSizeSet (uint8_t endpointAddress, uint8_t endpointSizeConfig)
  Sets the endpoint size for a default type IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutIsoSizeSet (uint8_t endpointAddress, uint8_t endpointSizeConfig)
  Sets the endpoint size for an isochronous OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInIsoSizeSet (uint8_t endpointAddress, uint8_t endpointSizeConfig)
  Sets the endpoint size for an isochronous IN endpoint.
- static ALWAYS_INLINE uint8_t USB_EndpointOutDefaultSizeGet (uint8_t endpointAddress)
  Gets the size of a default type OUT endpoint.
- static ALWAYS_INLINE uint8_t USB_EndpointInDefaultSizeGet (uint8_t endpointAddress)
  Gets the size of a default type IN endpoint.

- static ALWAYS_INLINE uint8_t USB_EndpointOutIsoSizeGet (uint8_t endpointAddress)
  Gets the size of an isochronous OUT endpoint.
- static ALWAYS_INLINE uint8_t USB_EndpointInIsoSizeGet (uint8_t endpointAddress)
  Gets the size of an isochronous IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutTransactionCompleteInterruptEnable (uint8_t endpointAddress)
  Enables transaction complete interrupt for the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInTransactionCompleteInterruptEnable (uint8_t endpointAddress)
  Enables transaction complete interrupt for the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutTransactionCompleteInterruptDisable (uint8_t endpointAddress)
  Disables transaction complete interrupt for the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInTransactionCompleteDisable (uint8_t endpointAddress)
  Disables transaction complete interrupt for the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutMultipktEnable (uint8_t endpointAddress)
  Enables multipacket for the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInMultipktEnable (uint8_t endpointAddress)
  Enables multipacket for the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutMultipktDisable (uint8_t endpointAddress)
  Disables multipacket for the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInMultipktDisable (uint8_t endpointAddress)
  Disables multipacket for the specified IN endpoint.
- static ALWAYS_INLINE bool USB_EndpointOutMultipktIsEnabled (uint8_t endpointAddress)
  Checks if multipacket is enabled on the specified OUT endpoint.
- static ALWAYS_INLINE bool USB_EndpointInMultipktIsEnabled (uint8_t endpointAddress)
  Checks if multipacket is enabled on the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutAzlpEnable (uint8_t endpointAddress)
  Enables Auto Zero Length Packet (AZLP) on the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInAlzpEnable (uint8_t endpointAddress)
  Enables AZLP on the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutAzlpDisable (uint8_t endpointAddress)
  Disables AZLP on the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInAzlpDisable (uint8_t endpointAddress)
  Disables AZLP on the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutStall (uint8_t endpointAddress)
  Stalls the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInStall (uint8_t endpointAddress)
  Stalls the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutStallClear (uint8_t endpointAddress)
  Stops stalling the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInStallClear (uint8_t endpointAddress)

Stops stalling the specified IN endpoint.

- static ALWAYS_INLINE bool USB_EndpointOutIsStalled (uint8_t endpointAddress)
  Checks if the specified OUT endpoint is stalled.
- static ALWAYS_INLINE bool USB_EndpointInIsStalled (uint8_t endpointAddress)
  Checks if the specified IN endpoint is stalled.
- static ALWAYS_INLINE void USB_EndpointOutStallAck (uint8_t endpointAddress)
  Acknowledges that an OUT endpoint is stalled and Clears the USB STATUS register.
- static ALWAYS_INLINE void USB_EndpointInStallAck (uint8_t endpointAddress)
  Acknowledges that an IN endpoint is stalled and Clears the USB STATUS register.
- static ALWAYS_INLINE void USB_EndpointOutNAKSet (uint8_t endpointAddress)
  Sets OUT endpoint status to NAK.
- static ALWAYS_INLINE void USB_EndpointInNAKSet (uint8_t endpointAddress)
  Sets IN endpoint status to NAK.
- static ALWAYS_INLINE void USB_EndpointOutNAKClear (uint8_t endpointAddress)
  Clears the USB NAK status from the OUT endpoint STATUS register.
- static ALWAYS_INLINE void USB_EndpointInNAKClear (uint8_t endpointAddress)
  Clears the USB NAK status from the IN endpoint STATUS register.
- static ALWAYS_INLINE bool USB_EndpointOutNAKIsSet (uint8_t endpointAddress)
  Checks the OUT endpoint STATUS register for the NAK status.
- static ALWAYS_INLINE bool USB_EndpointInNAKIsSet (uint8_t endpointAddress)
  Checks the OUT endpoint STATUS register for the NAK status.
- static ALWAYS_INLINE void USB_EndpointOutTransactionCompleteAck (uint8_t endpointAddress)
  Acknowledges the transaction complete status on a specified OUT endpoint and Clears the USB STATUS register.
- static ALWAYS_INLINE void USB_EndpointInTransactionCompleteAck (uint8_t endpointAddress)
  Acknowledges the transaction complete status on a specified IN endpoint and Clears the USB STATUS register.
- static ALWAYS_INLINE bool USB_EndpointOutTransactionIsComplete (uint8_t endpointAddress)
  Checks if the USB OUT endpoint has the Transaction Complete status.
- static ALWAYS_INLINE bool USB_EndpointInTransactionIsComplete (uint8_t endpointAddress)
  Checks if the USB IN endpoint has the Transaction Complete status.
- static ALWAYS_INLINE void USB_EndpointOutSetupReceivedAck (uint8_t endpointAddress)
  Acknowledges the Setup Received status on a specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInSetupCompleteAck (uint8_t endpointAddress)
  Acknowledges the Setup Received status on a specified IN endpoint.
- static ALWAYS_INLINE bool USB_EndpointOutSetupIsReceived (uint8_t endpointAddress)
  Checks if the USB OUT endpoint has the Setup Received status.
- static ALWAYS_INLINE bool USB_EndpointInSetupIsReceived (uint8_t endpointAddress)
  Checks if the USB IN endpoint has the Setup Received status.
- static ALWAYS_INLINE void USB_EndpointOutDataToggleSet (uint8_t endpointAddress)
  Sets OUT endpoint data toggle.
- static ALWAYS_INLINE void USB_EndpointInDataToggleSet (uint8_t endpointAddress)
  Sets IN endpoint data toggle.

- static ALWAYS_INLINE void USB_EndpointOutDataToggleClear (uint8_t endpointAddress)
  Clears OUT endpoint data toggle.
- static ALWAYS_INLINE void USB_EndpointInDataToggleClear (uint8_t endpointAddress)
  Clears IN endpoint data toggle.
- static ALWAYS_INLINE bool USB_EndpointOutDataToggleIsSet (uint8_t endpointAddress)
  Checks if data toggle is set on the specified OUT endpoint.
- static ALWAYS_INLINE bool USB_EndpointInDataToggleIsSet (uint8_t endpointAddress)
  Checks if data toggle is set on the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointOutBufferSet (uint8_t endpointAddress, uint8_t *bufAddress)
  Sets endpoint buffer OUT.
- static ALWAYS_INLINE void USB_EndpointInBufferSet (uint8_t endpointAddress, uint8_t *bufAddress)
  Sets endpoint buffer IN.
- static ALWAYS_INLINE void USB_NumberBytesToSendSet (uint8_t endpointAddress, uint16_t numberBytes)
  Sets how many bytes of data are intended to be sent from the specified endpoint.
- static ALWAYS_INLINE uint16_t USB_NumberBytesToSendGet (uint8_t endpointAddress)
  Reads out the CNT register to know how many bytes of data are intended to be sent from the specified endpoint.
- static ALWAYS_INLINE void USB_NumberBytesToSendReset (uint8_t endpointAddress)
  Clears the CNT register to tell the peripheral no data is intended to be sent from the specified endpoint.
- static ALWAYS_INLINE uint16_t USB_NumberBytesSentGet (uint8_t endpointAddress)
  Reads out how many bytes have been sent from the specified endpoint.
- static ALWAYS_INLINE void USB_NumberBytesSentReset (uint8_t endpointAddress)
  Clears the MCNT register that keeps track of how many bytes of data have been sent.
- static ALWAYS_INLINE void USB_NumberBytesToReceiveSet (uint8_t endpointAddress, uint16_t numberBytes)
  Sets how many bytes of data are expected to be received on a specified endpoint.
- static ALWAYS_INLINE uint16_t USB_NumberBytesToReceiveGet (uint8_t endpointAddress)
  Gets how many bytes of data are expected to be received on a specified endpoint.
- static ALWAYS_INLINE void USB_NumberBytesToReceiveReset (uint8_t endpointAddress)
  Clears the MCNT register to tell the peripheral no data is intended to be received on the specified endpoint.
- static ALWAYS_INLINE uint16_t USB_NumberBytesReceivedGet (uint8_t endpointAddress)
  Gets how many bytes of data have been received on a specified endpoint.
- static ALWAYS_INLINE void USB_NumberBytesReceivedReset (uint8_t endpointAddress)
  Resets the counter that counts amount of bytes of data received on a specific endpoint.
- static ALWAYS_INLINE bool USB_EndpointOutOverUnderflowIsSet (uint8_t endpointAddress)
  Checks if OUT endpoint has overflowed.
- static ALWAYS_INLINE bool USB_EndpointInOverUnderflowIsSet (uint8_t endpointAddress)
  Checks if IN endpoint has underflowed.
- static ALWAYS_INLINE void USB_EndpointOutOverUnderflowAck (uint8_t endpointAddress)

Acknowledges overflow on the specified OUT endpoint.

- static ALWAYS_INLINE void USB_EndpointInOverUnderflowAck (uint8_t endpointAddress)
  Acknowledges underflow on the specified IN endpoint.
- static ALWAYS_INLINE bool USB_EndpointOutCRCHasFailed (uint8_t endpointAddress)
  Checks if the specified OUT endpoint has a Cyclic Redundancy Check (CRC) failure.
- static ALWAYS_INLINE bool USB_EndpointInCRCHasFailed (uint8_t endpointAddress)
  Checks if the specified IN endpoint has a CRC failure.
- static ALWAYS_INLINE void USB_EndpointOutCRCAck (uint8_t endpointAddress)
  Acknowledges a CRC failure on the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInCRCAck (uint8_t endpointAddress)
  Acknowledges a CRC failure on the specified IN endpoint.
- static ALWAYS_INLINE void USB_GlobalNAKEnable (void)
  Enables global NAK.
- static ALWAYS_INLINE void USB_GlobalNAKDisable (void)
  Disables global NAK.
- static ALWAYS_INLINE bool USB_GlobalNAKIsEnable (void)
  Checks the global NAK setting.
- static ALWAYS_INLINE void USB_ConnectionAttach (void)
  Tells the USB peripheral to attach.
- static ALWAYS_INLINE void USB_ConnectionDetach (void)
  Tells the USB peripheral to detach.
- static ALWAYS_INLINE bool USB_ConnectionIsAttach (void)
  Checks if the USB connection is attached.
- static ALWAYS_INLINE void USB_Enable (void)
  Enables the USB peripheral.
- static ALWAYS_INLINE void USB_Disable (void)
  Disables the USB peripheral.
- static ALWAYS_INLINE bool USB_IsEnable (void)
  Checks if the USB peripheral is enabled.
- static ALWAYS_INLINE void USB_FifoEnable (void)
  Enables USB FIFO.
- static ALWAYS_INLINE void USB_FifoDisable (void)
  Disables USB FIFO.
- static ALWAYS_INLINE bool USB_FifoIsEnable (void)
  Checks if USB FIFO has been enabled.
- static ALWAYS_INLINE void USB_AutomaticGlobalNAKEnable (void)
  Enables automatic global NAK for the USB peripheral.
- static ALWAYS_INLINE void USB_AutomaticGlobalNAKDisable (void)
  Disables automatic global NAK for the USB peripheral.
- static ALWAYS_INLINE bool USB_AutomaticGlobalNAKIsEnable (void)
  Checks if automatic global NAK has been enabled.
- static ALWAYS_INLINE void USB_FrameNumEnable (void)

Enables storing the last SOF token frame number in FRAMENUM. This is a device-specific function.

- static ALWAYS_INLINE void USB_FrameNumDisable (void)
  Disables storing the last SOF token frame number in FRAMENUM. This is a device-specific function.
- static ALWAYS_INLINE bool USB_FrameNumIsEnable (void)
  Checks if storing of the last SOF token frame number is enabled. This is a device-specific function.
- static ALWAYS_INLINE void USB_MaxEndpointsSet (uint8_t maxEndpoint)
  Sets maximum number of endpoint addresses used by the USB peripheral.
- static ALWAYS_INLINE void USB_MaxEndpointsReset (void)
  Clears the USB endpoint maximum, setting the maximum endpoint to EP0.
- static ALWAYS_INLINE uint8_t USB_MaxEndpointsGet (void)
  Checks what the maximum number of endpoint addresses is.
- static ALWAYS_INLINE void USB_EndpointTableAddressSet (USB_EP_PAIR_t *endpointTableAddress)
  Sets the address of the endpoint table. This is a device-specific function.
- static ALWAYS_INLINE void USB_EndpointTableAddressReset (void)
  Sets the address of the endpoint table to 0. This is a device-specific function.
- static ALWAYS_INLINE uint16_t USB_EndpointTableAddressGet (void)
  Gets the address of the endpoint table. This is a device-specific function.
- static ALWAYS_INLINE void USB_FifoReadPointerReset (void)
  Resets the read FIFO pointer. This is a device-specific function.
- static ALWAYS_INLINE int8_t USB_FifoReadPointerGet (void)
  Gets the read FIFO pointer. This is a device-specific function.
- static ALWAYS_INLINE void USB_FifoWritePointerReset (void)
  Resets the write FIFO pointer. This is a device-specific function.
- static ALWAYS_INLINE int8_t USB_FifoWritePointerGet (void)
  Gets the write FIFO pointer. This is a device-specific function.
- static ALWAYS_INLINE void USB_DeviceAddressSet (uint8_t usbAddress)
  Sets the device address.
- static ALWAYS_INLINE void USB_DeviceAddressReset (void)
  Resets the device address.
- static ALWAYS_INLINE uint8_t USB_DeviceAddressGet (void)
  Gets the device address.
- static ALWAYS_INLINE void USB_UpstreamResumeEnable (void)
  Enables an upstream resume to be initated.
- static ALWAYS_INLINE bool USB_UpstreamResumeIsEnable (void)
  Checks if upstream resume is enabled, but not yet initiated.
- static ALWAYS_INLINE uint8_t USB_BusStateGet (void)
  Gets the USB bus state.
- static ALWAYS_INLINE bool USB_BusStateIs (uint8_t bus_state_bm)
  Checks if the USB bus has any specific status flags set.
- static ALWAYS_INLINE void USB_SOFInterruptEnable (void)

Enables the USB Start-Of-Frame interrupt.

- static ALWAYS_INLINE void USB_SOFInterruptDisable (void)
  Disables the USB Start-Of-Frame interrupt.
- static ALWAYS_INLINE void USB_SOFInterruptClear (void)
  Clears the USB Start-Of-Frame Interrupt flag.
- static ALWAYS_INLINE bool USB_SOFInterruptIs (void)
  Checks if the USB Start-Of-Frame interrupt has been triggered.
- static ALWAYS_INLINE void USB_SuspendInterruptEnable (void)
  Enables the USB Suspend interrupt.
- static ALWAYS_INLINE void USB_SuspendInterruptDisable (void)
  Disables the USB Suspend interrupt.
- static ALWAYS_INLINE void USB_SuspendInterruptClear (void)
  Clears the USB Suspend Interrupt flag.
- static ALWAYS_INLINE bool USB_SuspendInterruptIs (void)
  Checks if the USB Suspend interrupt has been triggered.
- static ALWAYS_INLINE void USB_ResumeInterruptEnable (void)
  Enables the USB Resume interrupt.
- static ALWAYS_INLINE void USB_ResumeInterruptDisable (void)
  Disables the USB Resume interrupt.
- static ALWAYS_INLINE void USB_ResumeInterruptClear (void)
  Clears the USB Resume Interrupt flag.
- static ALWAYS_INLINE bool USB_ResumeInterruptIs (void)
  Checks if the USB Resume interrupt has been triggered.
- static ALWAYS_INLINE void USB_ResetInterruptEnable (void)
  Enables the USB Reset interrupt.
- static ALWAYS_INLINE void USB_ResetInterruptDisable (void)
  Disables the USB Reset interrupt.
- static ALWAYS_INLINE void USB_ResetInterruptClear (void)
  Clears the USB Reset Interrupt flag.
- static ALWAYS_INLINE bool USB_ResetInterruptIs (void)
  Checks if the USB Reset interrupt has been triggered.
- static ALWAYS_INLINE void USB_StalledInterruptEnable (void)
  Enables the USB Stalled interrupt.
- static ALWAYS_INLINE void USB_StalledInterruptDisable (void)
  Disables the USB Stalled interrupt.
- static ALWAYS_INLINE void USB_StalledInterruptClear (void)
  Clears the USB Stalled Interrupt flag.
- static ALWAYS_INLINE bool USB_StalledInterruptIs (void)
  Checks if the USB Stalled interrupt has been triggered.
- static ALWAYS_INLINE void USB_UnderflowInterruptEnable (void)
  Enables the USB Underflow interrupt.
- static ALWAYS_INLINE void USB_UnderflowInterruptDisable (void)

Disables the USB Underflow interrupt.

- static ALWAYS_INLINE void USB_UnderflowInterruptClear (void)
  Clears the USB Underflow Interrupt flag.
- static ALWAYS_INLINE bool USB_UnderflowInterruptIs (void)
  Checks if an Underflow interrupt has been triggered.
- static ALWAYS_INLINE void USB_OverflowInterruptEnable (void)
  Enables the USB Overflow interrupt.
- static ALWAYS_INLINE void USB_OverflowInterruptDisable (void)
  Disables the USB Overflow interrupt.
- static ALWAYS_INLINE void USB_OverflowInterruptClear (void)
  Clears the USB Overflow Interrupt flag.
- static ALWAYS_INLINE bool USB_OverflowInterruptIs (void)
  Checks if an Overflow interrupt has been triggered.
- static ALWAYS_INLINE void USB_TransactionCompleteInterruptEnable (void)
  Enables the USB Transaction Complete interrupt.
- static ALWAYS_INLINE void USB_TransactionCompleteInterruptDisable (void)
  Disables the USB Transaction Complete interrupt.
- static ALWAYS_INLINE void USB_TransactionCompleteInterruptAck (void)
  Clears the USB Transaction Complete Interrupt flag.
- static ALWAYS_INLINE bool USB_TransactionCompleteInterruptIs (void)
  Checks if a Transaction Complete interrupt has been triggered.
- static ALWAYS_INLINE bool USB_ReadModifyWriteInterruptIs (void)
  Checks if the USB Read-Modify-Write Interrupt is enabled.
- static ALWAYS_INLINE void USB_GlobalNAKDoneInterruptEnable (void)
  Enables the USB Global NAK Done interrupt.
- static ALWAYS_INLINE void USB_GlobalNAKDoneInterruptDisable (void)
  Disables the USB Global NAK Done interrupt.
- static ALWAYS_INLINE void USB_GlobalNAKDoneInterruptAck (void)
  Clears the USB Global NAK Done Interrupt flag.
- static ALWAYS_INLINE bool USB_GlobalNAKDoneInterruptIs (void)
  Checks if the USB Global NAK Done interrupt has been triggered.
- static ALWAYS_INLINE void USB_SetupInterruptEnable (void)
  Enables the USB Setup interrupt.
- static ALWAYS_INLINE void USB_SetupInterruptDisable (void)
  Disables the USB Setup interrupt.
- static ALWAYS_INLINE void USB_SetupInterruptClear (void)
  Clears the USB Setup Interrupt flag.
- static ALWAYS_INLINE bool USB_SetupInterruptIs (void)
  Checks if a USB Setup interrupt has been triggered.
- static ALWAYS_INLINE void USB_InterruptFlagsClear (void)
  Clears all the USB Interrupt flags.

4.4.14.2 Definition Documentation

4.4.14.2.1 ALWAYS_INLINE

#define ALWAYS_INLINE __attribute__((always_inline)) inline

Alias that makes always inline function definitions more readable.

4.4.14.3 Function Documentation

4.4.14.3.1 USB_AutomaticGlobalNAKDisable()

static ALWAYS_INLINE void USB_AutomaticGlobalNAKDisable (void)[static]

Disables automatic global NAK for the USB peripheral.

Parameters:

None.

Returns:

None.

4.4.14.3.2 USB_AutomaticGlobalNAKEnable()

static ALWAYS_INLINE void USB_AutomaticGlobalNAKEnable (void)[static]

Enables automatic global NAK for the USB peripheral.

Parameters:

None.

Returns:

None.

4.4.14.3.3 USB_AutomaticGlobalNAKIsEnable()

static ALWAYS_INLINE bool USB_AutomaticGlobalNAKIsEnable (void)[static]

Checks if automatic global NAK has been enabled.

Parameters:

None.

Return values:

0	- Automatic global NAK is not enabled
1	- Automatic global NAK is enabled

4.4.14.3.4 USB_BusStateGet()

static ALWAYS_INLINE uint8_t USB_BusStateGet (void)[static]

Gets the USB bus state.

Parameters:

None.

Returns:

The state of the USB bus

4.4.14.3.5 USB_BusStateIs()

static ALWAYS_INLINE bool USB_BusStateIs (uint8_t bus_state_bm)[static]

Checks if the USB bus has any specific status flags set.

Parameters:

bus_state_bm	- The bitmap of the specific status flags to check

Return values:

0	- No status flags set
1	- The bus has one or more specified status flags set

4.4.14.3.6 USB_ConnectionAttach()

static ALWAYS_INLINE void USB_ConnectionAttach (void)[static]

Tells the USB peripheral to attach.

Parameters:

None.

Returns:

None.

4.4.14.3.7 USB_ConnectionDetach()

static ALWAYS_INLINE void USB_ConnectionDetach (void)[static]

Tells the USB peripheral to detach.

Parameters:

None.

Returns:

None.

4.4.14.3.8 USB_ConnectionIsAttach()

static ALWAYS_INLINE bool USB_ConnectionIsAttach (void)[static]

Checks if the USB connection is attached.

Parameters:

None.

Return values:

0	- USB connection is not attached
1	- USB connection is attached

4.4.14.3.9 USB_DeviceAddressGet()

static ALWAYS_INLINE uint8_t USB_DeviceAddressGet (void)[static]

Gets the device address.

Parameters:

None.

Returns:

The device address

4.4.14.3.10 USB_DeviceAddressReset()

static ALWAYS_INLINE void USB_DeviceAddressReset (void)[static]

Resets the device address.

Parameters:

None.

Returns:

None.

4.4.14.3.11 USB_DeviceAddressSet()

static ALWAYS_INLINE void USB_DeviceAddressSet (uint8_t usbAddress)[static]

Sets the device address.

Parameters:

usbAddress	- The device address to set

Returns:

None.

4.4.14.3.12 USB_Disable()

static ALWAYS_INLINE void USB_Disable (void)[static]

Disables the USB peripheral.

Parameters:

None.

Returns:

None.

4.4.14.3.13 USB_Enable()

static ALWAYS_INLINE void USB_Enable (void)[static]

Enables the USB peripheral.

Parameters:

None.

Returns:

None.

4.4.14.3.14 USB_EndpointInAlzpEnable()

static ALWAYS_INLINE void USB_EndpointInAlzpEnable (uint8_t endpointAddress)[static]

Enables AZLP on the specified IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.15 USB_EndpointInAzlpDisable()

static ALWAYS_INLINE void USB_EndpointInAzlpDisable (uint8_t endpointAddress)[static]

Disables AZLP on the specified IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.16 USB_EndpointInBufferSet()

static ALWAYS_INLINE void USB_EndpointInBufferSet (uint8_t endpointAddress, uint8_t * bufAddress)[static]

Sets endpoint buffer IN.

Parameters:

endpointAddress	- Address of the endpoint
bufAddress	- Address pointer to buffer

Returns:

None.

MISRA C:2012 Deviation **Advisory:** misra-c2012- 11.4
Justification: A conversion should not be performed between a pointer to object and an integer type. The EP.IN.DATAPTR register is a 16-bit register, expecting an AVR DU specific 16-bit RAM address.

4.4.14.3.17 USB_EndpointInControlSet()

static ALWAYS_INLINE void USB_EndpointInControlSet (uint8_t endpointAddress, uint8_t value) [static]

Sets endpoint control IN.

Parameters:

endpointAddress	- Address of the endpoint
value	- Register bitmask

Returns:

None.

4.4.14.3.18 USB_EndpointInCRCAck()

static ALWAYS_INLINE void USB_EndpointInCRCAck (uint8_t endpointAddress)[static]

Acknowledges a CRC failure on the specified IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.19 USB_EndpointInCRCHasFailed()

static ALWAYS_INLINE bool USB_EndpointInCRCHasFailed (uint8_t endpointAddress)[static]

Checks if the specified IN endpoint has a CRC failure.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- No CRC failure on endpoint
1	- CRC failure on endpoint

4.4.14.3.20 USB_EndpointInDataToggleClear()

static ALWAYS_INLINE void USB_EndpointInDataToggleClear (uint8_t endpointAddress)[static]

Clears IN endpoint data toggle.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.21 USB_EndpointInDataToggleIsSet()

static ALWAYS_INLINE bool USB_EndpointInDataToggleIsSet (uint8_t endpointAddress)[static]

Checks if data toggle is set on the specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- Data toggle is not set
1	- Data toggle is set

4.4.14.3.22 USB_EndpointInDataToggleSet()

static ALWAYS_INLINE void USB_EndpointInDataToggleSet (uint8_t endpointAddress)[static]

Sets IN endpoint data toggle.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.23 USB_EndpointInDefaultSizeGet()

static ALWAYS_INLINE uint8_t USB_EndpointInDefaultSizeGet (uint8_t endpointAddress)[static]

Gets the size of a default type IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

IN endpoint size

4.4.14.3.24 USB_EndpointInDefaultSizeSet()

static ALWAYS_INLINE void USB_EndpointInDefaultSizeSet (uint8_t endpointAddress, uint8_t endpointSizeConfig)[static]

Sets the endpoint size for a default type IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint
endpointSizeConfig	- Size of endpoint to set

Returns:

None.

4.4.14.3.25 USB_EndPointInDisable()

static ALWAYS_INLINE void USB_EndPointInDisable (uint8_t endpointAddress)[static]

Disables the IN endpoint with the given address.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.26 USB_EndPointInIsEnabled()

static ALWAYS_INLINE bool USB_EndPointInIsEnabled (uint8_t endpointAddress)[static]

Checks if the IN endpoint at the given address is enabled.

Parameters:

None.

Return values:

0	Endpoint not enabled
1	Endpoint enabled

4.4.14.3.27 USB_EndpointInIsoSizeGet()

static ALWAYS_INLINE uint8_t USB_EndpointInIsoSizeGet (uint8_t endpointAddress)[static]

Gets the size of an isochronous IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

IN endpoint size

4.4.14.3.28 USB_EndpointInIsoSizeSet()

static ALWAYS_INLINE void USB_EndpointInIsoSizeSet (uint8_t endpointAddress, uint8_t endpointSizeConfig)[static]

Sets the endpoint size for an isochronous IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint
endpointSizeConfig	- Size of endpoint to set

Returns:

None.

4.4.14.3.29 USB_EndpointInIsStalled()

static ALWAYS_INLINE bool USB_EndpointInIsStalled (uint8_t endpointAddress)[static]

Checks if the specified IN endpoint is stalled.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- Endpoint is not stalled
1	- Endpoint is stalled

4.4.14.3.30 USB_EndpointInMultipktDisable()

static ALWAYS_INLINE void USB_EndpointInMultipktDisable (uint8_t endpointAddress)[static]

Disables multipacket for the specified IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.31 USB_EndpointInMultipktEnable()

static ALWAYS_INLINE void USB_EndpointInMultipktEnable (uint8_t endpointAddress)[static]

Enables multipacket for the specified IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.32 USB_EndpointInMultipktIsEnabled()

static ALWAYS_INLINE bool USB_EndpointInMultipktIsEnabled (uint8_t endpointAddress)[static]

Checks if multipacket is enabled on the specified IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- Multipacket disabled
1	- Multipacket enabled

4.4.14.3.33 USB_EndpointInNAKClear()

static ALWAYS_INLINE void USB_EndpointInNAKClear (uint8_t endpointAddress)[static]

Clears the USB NAK status from the IN endpoint STATUS register.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.34 USB_EndpointInNAKIsSet()

static ALWAYS_INLINE bool USB_EndpointInNAKIsSet (uint8_t endpointAddress)[static]

Checks the OUT endpoint STATUS register for the NAK status.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- Endpoint is not NAKed
1	- Endpoint is NAKed

4.4.14.3.35 USB_EndpointInNAKSet()

static ALWAYS_INLINE void USB_EndpointInNAKSet (uint8_t endpointAddress)[static]

Sets IN endpoint status to NAK.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.36 USB_EndpointInOverUnderflowAck()

static ALWAYS_INLINE void USB_EndpointInOverUnderflowAck (uint8_t endpointAddress)[static]

Acknowledges underflow on the specified IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.37 USB_EndpointInOverUnderflowIsSet()

static ALWAYS_INLINE bool USB_EndpointInOverUnderflowIsSet (uint8_t endpointAddress)[static]

Checks if IN endpoint has underflowed.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- No underflow on endpoint
1	- Undeflow on endpoint

4.4.14.3.38 USB_EndpointInSetupCompleteAck()

static ALWAYS_INLINE void USB_EndpointInSetupCompleteAck (uint8_t endpointAddress)[static]

Acknowledges the Setup Received status on a specified IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.39 USB_EndpointInSetupIsReceived()

static ALWAYS_INLINE bool USB_EndpointInSetupIsReceived (uint8_t endpointAddress)[static]

Checks if the USB IN endpoint has the Setup Received status.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- No setup received
1	- Setup received

4.4.14.3.40 USB_EndpointInStall()

static ALWAYS_INLINE void USB_EndpointInStall (uint8_t endpointAddress)[static]

Stalls the specified IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.41 USB_EndpointInStallAck()

static ALWAYS_INLINE void USB_EndpointInStallAck (uint8_t endpointAddress)[static]

Acknowledges that an IN endpoint is stalled and Clears the USB STATUS register.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.42 USB_EndpointInStallClear()

static ALWAYS_INLINE void USB_EndpointInStallClear (uint8_t endpointAddress)[static]

Stops stalling the specified IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.43 USB_EndpointInStatusClear()

static ALWAYS_INLINE void USB_EndpointInStatusClear (uint8_t endpointAddress)[static]

Clears IN endpoint status.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.44 USB_EndpointInTransactionCompleteAck()

static ALWAYS_INLINE void USB_EndpointInTransactionCompleteAck (uint8_t endpointAddress) [static]

Acknowledges the transaction complete status on a specified IN endpoint and Clears the USB STATUS register.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.45 USB_EndpointInTransactionCompleteDisable()

static ALWAYS_INLINE void USB_EndpointInTransactionCompleteDisable (uint8_t endpointAddress) [static]

Disables transaction complete interrupt for the specified IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.46 USB_EndpointInTransactionCompleteInterruptEnable()

static ALWAYS_INLINE void USB_EndpointInTransactionCompleteInterruptEnable (uint8_t endpointAddress)[static]

Enables transaction complete interrupt for the specified IN endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.47 USB_EndpointInTransactionIsComplete()

static ALWAYS_INLINE bool USB_EndpointInTransactionIsComplete (uint8_t endpointAddress)[static]

Checks if the USB IN endpoint has the Transaction Complete status.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- Transaction incomplete
1	- Transaction complete

4.4.14.3.48 USB_EndPointInTypeConfigGet()

static ALWAYS_INLINE uint8_t USB_EndPointInTypeConfigGet (uint8_t endpointAddress)[static]

Gets the IN endpoint configuration at the given address.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

Endpoint configuration type

4.4.14.3.49 USB_EndpointOutAzlpDisable()

static ALWAYS_INLINE void USB_EndpointOutAzlpDisable (uint8_t endpointAddress)[static]

Disables AZLP on the specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.50 USB_EndpointOutAzlpEnable()

static ALWAYS_INLINE void USB_EndpointOutAzlpEnable (uint8_t endpointAddress)[static]

Enables Auto Zero Length Packet (AZLP) on the specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.51 USB_EndpointOutBufferSet()

static ALWAYS_INLINE void USB_EndpointOutBufferSet (uint8_t endpointAddress, uint8_t * bufAddress)[static]

Sets endpoint buffer OUT.

Parameters:

endpointAddress	- Address of the endpoint
bufAddress	- Address pointer to buffer

Returns:

None.

MISRA C:2012 Deviation **Advisory:** misra-c2012- 11.4

Justification: A conversion should not be performed between a pointer to object and an integer type. The EP.OUT.DATAPTR register is a 16-bit register, expecting an AVR DU specific 16-bit RAM address.

4.4.14.3.52 USB_EndpointOutControlSet()

static ALWAYS_INLINE void USB_EndpointOutControlSet (uint8_t endpointAddress, uint8_t value) [static]

Sets endpoint control OUT.

Parameters:

endpointAddress	- Address of the endpoint
value	- Register bitmask

Returns:

None.

4.4.14.3.53 USB_EndpointOutCRCAck()

static ALWAYS_INLINE void USB_EndpointOutCRCAck (uint8_t endpointAddress)[static]

Acknowledges a CRC failure on the specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.54 USB_EndpointOutCRCHasFailed()

static ALWAYS_INLINE bool USB_EndpointOutCRCHasFailed (uint8_t endpointAddress)[static]

Checks if the specified OUT endpoint has a Cyclic Redundancy Check (CRC) failure.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- No CRC failure on endpoint
1	- CRC failure on endpoint

4.4.14.3.55 USB_EndpointOutDataToggleClear()

static ALWAYS_INLINE void USB_EndpointOutDataToggleClear (uint8_t endpointAddress)[static]

Clears OUT endpoint data toggle.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.56 USB_EndpointOutDataToggleIsSet()

static ALWAYS_INLINE bool USB_EndpointOutDataToggleIsSet (uint8_t endpointAddress)[static]

Checks if data toggle is set on the specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- Data toggle is not set
1	- Data toggle is set

4.4.14.3.57 USB_EndpointOutDataToggleSet()

static ALWAYS_INLINE void USB_EndpointOutDataToggleSet (uint8_t endpointAddress)[static]

Sets OUT endpoint data toggle.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.58 USB_EndpointOutDefaultSizeGet()

static ALWAYS_INLINE uint8_t USB_EndpointOutDefaultSizeGet (uint8_t endpointAddress)[static]

Gets the size of a default type OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

OUT endpoint size

4.4.14.3.59 USB_EndpointOutDefaultSizeSet()

static ALWAYS_INLINE void USB_EndpointOutDefaultSizeSet (uint8_t endpointAddress, uint8_t endpointSizeConfig)[static]

Sets the endpoint size for a default type OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint
endpointSizeConfig	- Size of endpoint to set

Returns:

None.

4.4.14.3.60 USB_EndPointOutDisable()

static ALWAYS_INLINE void USB_EndPointOutDisable (uint8_t endpointAddress)[static]

Disables the OUT endpoint with the given address.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.61 USB_EndPointOutIsEnabled()

static ALWAYS_INLINE bool USB_EndPointOutIsEnabled (uint8_t endpointAddress)[static]

Checks if the OUT endpoint at the given address is enabled.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	Endpoint not enabled
1	Endpoint enabled

4.4.14.3.62 USB_EndpointOutIsoSizeGet()

static ALWAYS_INLINE uint8_t USB_EndpointOutIsoSizeGet (uint8_t endpointAddress)[static]

Gets the size of an isochronous OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

OUT endpoint size

4.4.14.3.63 USB_EndpointOutIsoSizeSet()

static ALWAYS_INLINE void USB_EndpointOutIsoSizeSet (uint8_t endpointAddress, uint8_t endpointSizeConfig)[static]

Sets the endpoint size for an isochronous OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint
endpointSizeConfig	- Size of endpoint to set

Returns:

None.

4.4.14.3.64 USB_EndpointOutIsStalled()

static ALWAYS_INLINE bool USB_EndpointOutIsStalled (uint8_t endpointAddress)[static]

Checks if the specified OUT endpoint is stalled.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- Endpoint is not stalled
1	- Endpoint is stalled

4.4.14.3.65 USB_EndpointOutMultipktDisable()

static ALWAYS_INLINE void USB_EndpointOutMultipktDisable (uint8_t endpointAddress)[static]

Disables multipacket for the specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.66 USB_EndpointOutMultipktEnable()

static ALWAYS_INLINE void USB_EndpointOutMultipktEnable (uint8_t endpointAddress)[static]

Enables multipacket for the specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.67 USB_EndpointOutMultipktIsEnabled()

static ALWAYS_INLINE bool USB_EndpointOutMultipktIsEnabled (uint8_t endpointAddress)[static]

Checks if multipacket is enabled on the specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- Multipacket disabled
1	- Multipacket enabled

4.4.14.3.68 USB_EndpointOutNAKClear()

static ALWAYS_INLINE void USB_EndpointOutNAKClear (uint8_t endpointAddress)[static]

Clears the USB NAK status from the OUT endpoint STATUS register.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.69 USB_EndpointOutNAKIsSet()

static ALWAYS_INLINE bool USB_EndpointOutNAKIsSet (uint8_t endpointAddress)[static]

Checks the OUT endpoint STATUS register for the NAK status.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- Endpoint is not NAKed
1	- Endpoint is NAKed

4.4.14.3.70 USB_EndpointOutNAKSet()

static ALWAYS_INLINE void USB_EndpointOutNAKSet (uint8_t endpointAddress)[static]

Sets OUT endpoint status to NAK.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.71 USB_EndpointOutOverUnderflowAck()

static ALWAYS_INLINE void USB_EndpointOutOverUnderflowAck (uint8_t endpointAddress)[static]

Acknowledges overflow on the specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.72 USB_EndpointOutOverUnderflowIsSet()

static ALWAYS_INLINE bool USB_EndpointOutOverUnderflowIsSet (uint8_t endpointAddress)[static]

Checks if OUT endpoint has overflowed.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- No overflow on endpoint
1	- Overflow on endpoint

4.4.14.3.73 USB_EndpointOutSetupIsReceived()

static ALWAYS_INLINE bool USB_EndpointOutSetupIsReceived (uint8_t endpointAddress)[static]

Checks if the USB OUT endpoint has the Setup Received status.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- No setup received
1	- Setup received

4.4.14.3.74 USB_EndpointOutSetupReceivedAck()

static ALWAYS_INLINE void USB_EndpointOutSetupReceivedAck (uint8_t endpointAddress)[static]

Acknowledges the Setup Received status on a specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.75 USB_EndpointOutStall()

static ALWAYS_INLINE void USB_EndpointOutStall (uint8_t endpointAddress)[static]

Stalls the specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.76 USB_EndpointOutStallAck()

static ALWAYS_INLINE void USB_EndpointOutStallAck (uint8_t endpointAddress)[static]

Acknowledges that an OUT endpoint is stalled and Clears the USB STATUS register.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.77 USB_EndpointOutStallClear()

static ALWAYS_INLINE void USB_EndpointOutStallClear (uint8_t endpointAddress)[static]

Stops stalling the specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.78 USB_EndpointOutStatusClear()

static ALWAYS_INLINE void USB_EndpointOutStatusClear (uint8_t endpointAddress)[static]

Clears OUT endpoint status.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.79 USB_EndpointOutTransactionCompleteAck()

static ALWAYS_INLINE void USB_EndpointOutTransactionCompleteAck (uint8_t endpointAddress) [static]

Acknowledges the transaction complete status on a specified OUT endpoint and Clears the USB STATUS register.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.80 USB_EndpointOutTransactionCompleteInterruptDisable()

static ALWAYS_INLINE void USB_EndpointOutTransactionCompleteInterruptDisable (uint8_t endpointAddress)[static]

Disables transaction complete interrupt for the specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.81 USB_EndpointOutTransactionCompleteInterruptEnable()

static ALWAYS_INLINE void USB_EndpointOutTransactionCompleteInterruptEnable (uint8_t endpointAddress)[static]

Enables transaction complete interrupt for the specified OUT endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.82 USB_EndpointOutTransactionIsComplete()

static ALWAYS_INLINE bool USB_EndpointOutTransactionIsComplete (uint8_t endpointAddress) [static]

Checks if the USB OUT endpoint has the Transaction Complete status.

Parameters:

endpointAddress	- Address of the endpoint

Return values:

0	- Transaction incomplete
1	- Transaction complete

4.4.14.3.83 USB_EndPointOutTypeConfigGet()

static ALWAYS_INLINE uint8_t USB_EndPointOutTypeConfigGet (uint8_t endpointAddress)[static]

Gets the OUT endpoint configuration at the given address.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

Endpoint configuration type

4.4.14.3.84 USB_EndpointTableAddressGet()

static ALWAYS_INLINE uint16_t USB_EndpointTableAddressGet (void)[static]

Gets the address of the endpoint table. This is a device-specific function.

Parameters:

None.

Returns:

The address of the endpoint table

4.4.14.3.85 USB_EndpointTableAddressReset()

static ALWAYS_INLINE void USB_EndpointTableAddressReset (void)[static]

Sets the address of the endpoint table to 0. This is a device-specific function.

Parameters:

None.

Returns:

None.

4.4.14.3.86 USB_EndpointTableAddressSet()

static ALWAYS_INLINE void USB_EndpointTableAddressSet (USB_EP_PAIR_t * endpointTableAddress) [static]

Sets the address of the endpoint table. This is a device-specific function.

Parameters:

endpointTableAddress	- Address of the endpoint table

Returns:

None.

4.4.14.3.87 USB_FifoDisable()

static ALWAYS_INLINE void USB_FifoDisable (void)[static]

Disables USB FIFO.

This is a device-specific function.

Parameters:

None.

Returns:

None.

4.4.14.3.88 USB_FifoEnable()

static ALWAYS_INLINE void USB_FifoEnable (void)[static]

Enables USB FIFO.

This is a device-specific function.

Parameters:

None.

Returns:

None.

4.4.14.3.89 USB_FifoIsEnable()

static ALWAYS_INLINE bool USB_FifoIsEnable (void)[static]

Checks if USB FIFO has been enabled.

This is a device-specific function.

Parameters:

None.

Return values:

0	- USB FIFO is not enabled
1	- USB FIFO is enabled

4.4.14.3.90 USB_FifoReadPointerGet()

static ALWAYS_INLINE int8_t USB_FifoReadPointerGet (void)[static]

Gets the read FIFO pointer. This is a device-specific function.

Parameters:

None.

Returns:

The FIFO read pointer

4.4.14.3.91 USB_FifoReadPointerReset()

static ALWAYS_INLINE void USB_FifoReadPointerReset (void)[static]

Resets the read FIFO pointer. This is a device-specific function.

Parameters:

None.

Returns:

None.

4.4.14.3.92 USB_FifoWritePointerGet()

static ALWAYS_INLINE int8_t USB_FifoWritePointerGet (void)[static]

Gets the write FIFO pointer. This is a device-specific function.

Parameters:

None.

Returns:

The FIFO write pointer

4.4.14.3.93 USB_FifoWritePointerReset()

static ALWAYS_INLINE void USB_FifoWritePointerReset (void)[static]

Resets the write FIFO pointer. This is a device-specific function.

Parameters:

None.

Returns:

None.

4.4.14.3.94 USB_FrameNumDisable()

static ALWAYS_INLINE void USB_FrameNumDisable (void)[static]

Disables storing the last SOF token frame number in FRAMENUM. This is a device-specific function.

Parameters:

None.

Returns:

None.

4.4.14.3.95 USB_FrameNumEnable()

static ALWAYS_INLINE void USB_FrameNumEnable (void)[static]

Enables storing the last SOF token frame number in FRAMENUM. This is a device-specific function.

Parameters:

None.

Returns:

None.

4.4.14.3.96 USB_FrameNumIsEnable()

static ALWAYS_INLINE bool USB_FrameNumIsEnable (void)[static]

Checks if storing of the last SOF token frame number is enabled. This is a device-specific function.

Parameters:

None.

Return values:

0	- Storing the last SOF token frame number in FRAMENUM is disabled
1	- Storing the last SOF token frame number in FRAMENUM is enabled

4.4.14.3.97 USB_GlobalNAKDisable()

static ALWAYS_INLINE void USB_GlobalNAKDisable (void)[static]

Disables global NAK.

Parameters:

None.

Returns:

None.

4.4.14.3.98 USB_GlobalNAKDoneInterruptAck()

static ALWAYS_INLINE void USB_GlobalNAKDoneInterruptAck (void)[static]

Clears the USB Global NAK Done Interrupt flag.

Parameters:

None.

Returns:

None.

4.4.14.3.99 USB_GlobalNAKDoneInterruptDisable()

static ALWAYS_INLINE void USB_GlobalNAKDoneInterruptDisable (void)[static]

Disables the USB Global NAK Done interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.100 USB_GlobalNAKDoneInterruptEnable()

static ALWAYS_INLINE void USB_GlobalNAKDoneInterruptEnable (void)[static]

Enables the USB Global NAK Done interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.101 USB_GlobalNAKDoneInterruptIs()

static ALWAYS_INLINE bool USB_GlobalNAKDoneInterruptIs (void)[static]

Checks if the USB Global NAK Done interrupt has been triggered.

Parameters:

None.

Return values:

0	- Interrupt not triggered
1	- Interrupt triggered

4.4.14.3.102 USB_GlobalNAKEnable()

static ALWAYS_INLINE void USB_GlobalNAKEnable (void)[static]

Enables global NAK.

Parameters:

None.

Returns:

None.

4.4.14.3.103 USB_GlobalNAKIsEnable()

static ALWAYS_INLINE bool USB_GlobalNAKIsEnable (void)[static]

Checks the global NAK setting.

Parameters:

None.

Return values:

0	- Global NAK is disabled
1	- Global NAK is enabled

4.4.14.3.104 USB_InterruptFlagsClear()

static ALWAYS_INLINE void USB_InterruptFlagsClear (void)[static]

Clears all the USB Interrupt flags.

Parameters:

None.

Returns:

None.

4.4.14.3.105 USB_IsEnable()

static ALWAYS_INLINE bool USB_IsEnable (void)[static]

Checks if the USB peripheral is enabled.

Parameters:

None.

Return values:

0	- USB peripheral not enabled
1	- USB peripheral enabled

4.4.14.3.106 USB_MaxEndpointsGet()

static ALWAYS_INLINE uint8_t USB_MaxEndpointsGet (void)[static]

Checks what the maximum number of endpoint addresses is.

Parameters:

None.

Returns:

Last accessible endpoint

4.4.14.3.107 USB_MaxEndpointsReset()

static ALWAYS_INLINE void USB_MaxEndpointsReset (void)[static]

Clears the USB endpoint maximum, setting the maximum endpoint to EP0.

Parameters:

None.

Returns:

None.

4.4.14.3.108 USB_MaxEndpointsSet()

static ALWAYS_INLINE void USB_MaxEndpointsSet (uint8_t maxEndpoint)[static]

Sets maximum number of endpoint addresses used by the USB peripheral.

Parameters:

maxEndpoint	- Last accessible endpoint

Returns:

None.

4.4.14.3.109 USB_NumberBytesReceivedGet()

static ALWAYS_INLINE uint16_t USB_NumberBytesReceivedGet (uint8_t endpointAddress)[static]

Gets how many bytes of data have been received on a specified endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

Amount of bytes expected

4.4.14.3.110 USB_NumberBytesReceivedReset()

static ALWAYS_INLINE void USB_NumberBytesReceivedReset (uint8_t endpointAddress)[static]

Resets the counter that counts amount of bytes of data received on a specific endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.111 USB_NumberBytesSentGet()

static ALWAYS_INLINE uint16_t USB_NumberBytesSentGet (uint8_t endpointAddress)[static]

Reads out how many bytes have been sent from the specified endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

Amount of bytes that have been sent

4.4.14.3.112 USB_NumberBytesSentReset()

static ALWAYS_INLINE void USB_NumberBytesSentReset (uint8_t endpointAddress)[static]

Clears the MCNT register that keeps track of how many bytes of data have been sent.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.113 USB_NumberBytesToReceiveGet()

static ALWAYS_INLINE uint16_t USB_NumberBytesToReceiveGet (uint8_t endpointAddress)[static]

Gets how many bytes of data are expected to be received on a specified endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

Amount of bytes expected

4.4.14.3.114 USB_NumberBytesToReceiveReset()

static ALWAYS_INLINE void USB_NumberBytesToReceiveReset (uint8_t endpointAddress)[static]

Clears the MCNT register to tell the peripheral no data is intended to be received on the specified endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.115 USB_NumberBytesToReceiveSet()

static ALWAYS_INLINE void USB_NumberBytesToReceiveSet (uint8_t endpointAddress, uint16_t numberBytes)[static]

Sets how many bytes of data are expected to be received on a specified endpoint.

Parameters:

endpointAddress	- Address of the endpoint
numberBytes	- The amount of bytes to receive

Returns:

None.

4.4.14.3.116 USB_NumberBytesToSendGet()

static ALWAYS_INLINE uint16_t USB_NumberBytesToSendGet (uint8_t endpointAddress)[static]

Reads out the CNT register to know how many bytes of data are intended to be sent from the specified endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

Amount of bytes to send

4.4.14.3.117 USB_NumberBytesToSendReset()

static ALWAYS_INLINE void USB_NumberBytesToSendReset (uint8_t endpointAddress)[static]

Clears the CNT register to tell the peripheral no data is intended to be sent from the specified endpoint.

Parameters:

endpointAddress	- Address of the endpoint

Returns:

None.

4.4.14.3.118 USB_NumberBytesToSendSet()

static ALWAYS_INLINE void USB_NumberBytesToSendSet (uint8_t endpointAddress, uint16_t numberBytes)[static]

Sets how many bytes of data are intended to be sent from the specified endpoint.

Parameters:

endpointAddress	- Address of the endpoint
numberBytes	- Amount of bytes to send

Returns:

None.

4.4.14.3.119 USB_OverflowInterruptClear()

static ALWAYS_INLINE void USB_OverflowInterruptClear (void)[static]

Clears the USB Overflow Interrupt flag.

Parameters:

None.

Returns:

None.

4.4.14.3.120 USB_OverflowInterruptDisable()

static ALWAYS_INLINE void USB_OverflowInterruptDisable (void)[static]

Disables the USB Overflow interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.121 USB_OverflowInterruptEnable()

static ALWAYS_INLINE void USB_OverflowInterruptEnable (void)[static]

Enables the USB Overflow interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.122 USB_OverflowInterruptIs()

static ALWAYS_INLINE bool USB_OverflowInterruptIs (void)[static]

Checks if an Overflow interrupt has been triggered.

Parameters:

None.

Return values:

0	- Interrupt not triggered
1	- Interrupt triggered

4.4.14.3.123 USB_ReadModifyWriteInterruptIs()

static ALWAYS_INLINE bool USB_ReadModifyWriteInterruptIs (void)[static]

Checks if the USB Read-Modify-Write Interrupt is enabled.

Parameters:

None.

Return values:

0	- Interrupt not enabled
1	- Interrupt enabled

4.4.14.3.124 USB_ResetInterruptClear()

static ALWAYS_INLINE void USB_ResetInterruptClear (void)[static]

Clears the USB Reset Interrupt flag.

Parameters:

None.

Returns:

None.

4.4.14.3.125 USB_ResetInterruptDisable()

static ALWAYS_INLINE void USB_ResetInterruptDisable (void)[static]

Disables the USB Reset interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.126 USB_ResetInterruptEnable()

static ALWAYS_INLINE void USB_ResetInterruptEnable (void)[static]

Enables the USB Reset interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.127 USB_ResetInterruptIs()

static ALWAYS_INLINE bool USB_ResetInterruptIs (void)[static]

Checks if the USB Reset interrupt has been triggered.

Parameters:

None.

Return values:

0	- Interrupt not triggered
1	- Interrupt triggered

4.4.14.3.128 USB_ResumeInterruptClear()

static ALWAYS_INLINE void USB_ResumeInterruptClear (void)[static]

Clears the USB Resume Interrupt flag.

Parameters:

None.

Returns:

None.

4.4.14.3.129 USB_ResumeInterruptDisable()

static ALWAYS_INLINE void USB_ResumeInterruptDisable (void)[static]

Disables the USB Resume interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.130 USB_ResumeInterruptEnable()

static ALWAYS_INLINE void USB_ResumeInterruptEnable (void)[static]

Enables the USB Resume interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.131 USB_ResumeInterruptIs()

static ALWAYS_INLINE bool USB_ResumeInterruptIs (void)[static]

Checks if the USB Resume interrupt has been triggered.

Parameters:

None.

Return values:

0	- Interrupt not triggered
1	- Interrupt triggered

4.4.14.3.132 USB_SetupInterruptClear()

static ALWAYS_INLINE void USB_SetupInterruptClear (void)[static]

Clears the USB Setup Interrupt flag.

Parameters:

None.

Returns:

None.

4.4.14.3.133 USB_SetupInterruptDisable()

static ALWAYS_INLINE void USB_SetupInterruptDisable (void)[static]

Disables the USB Setup interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.134 USB_SetupInterruptEnable()

static ALWAYS_INLINE void USB_SetupInterruptEnable (void)[static]

Enables the USB Setup interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.135 USB_SetupInterruptIs()

static ALWAYS_INLINE bool USB_SetupInterruptIs (void)[static]

Checks if a USB Setup interrupt has been triggered.

Parameters:

None.

Return values:

0	- Interrupt not triggered
1	- Interrupt triggered

4.4.14.3.136 USB_SOFInterruptClear()

static ALWAYS_INLINE void USB_SOFInterruptClear (void)[static]

Clears the USB Start-Of-Frame Interrupt flag.

Parameters:

None.

Returns:

None.

4.4.14.3.137 USB_SOFInterruptDisable()

static ALWAYS_INLINE void USB_SOFInterruptDisable (void)[static]

Disables the USB Start-Of-Frame interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.138 USB_SOFInterruptEnable()

static ALWAYS_INLINE void USB_SOFInterruptEnable (void)[static]

Enables the USB Start-Of-Frame interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.139 USB_SOFInterruptIs()

static ALWAYS_INLINE bool USB_SOFInterruptIs (void)[static]

Checks if the USB Start-Of-Frame interrupt has been triggered.

Parameters:

None.

Return values:

0	- Interrupt not triggered
1	- Interrupt triggered

4.4.14.3.140 USB_StalledInterruptClear()

static ALWAYS_INLINE void USB_StalledInterruptClear (void)[static]

Clears the USB Stalled Interrupt flag.

Parameters:

None.

Returns:

None.

4.4.14.3.141 USB_StalledInterruptDisable()

static ALWAYS_INLINE void USB_StalledInterruptDisable (void)[static]

Disables the USB Stalled interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.142 USB_StalledInterruptEnable()

static ALWAYS_INLINE void USB_StalledInterruptEnable (void)[static]

Enables the USB Stalled interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.143 USB_StalledInterruptIs()

static ALWAYS_INLINE bool USB_StalledInterruptIs (void)[static]

Checks if the USB Stalled interrupt has been triggered.

Parameters:

None.

Return values:

0	- Interrupt not triggered
1	- Interrupt triggered

4.4.14.3.144 USB_SuspendInterruptClear()

static ALWAYS_INLINE void USB_SuspendInterruptClear (void)[static]

Clears the USB Suspend Interrupt flag.

Parameters:

None.

Returns:

None.

4.4.14.3.145 USB_SuspendInterruptDisable()

static ALWAYS_INLINE void USB_SuspendInterruptDisable (void)[static]

Disables the USB Suspend interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.146 USB_SuspendInterruptEnable()

static ALWAYS_INLINE void USB_SuspendInterruptEnable (void)[static]

Enables the USB Suspend interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.147 USB_SuspendInterruptIs()

static ALWAYS_INLINE bool USB_SuspendInterruptIs (void)[static]

Checks if the USB Suspend interrupt has been triggered.

Parameters:

None.

Return values:

0	- Interrupt not triggered
1	- Interrupt triggered

4.4.14.3.148 USB_TransactionCompleteInterruptAck()

static ALWAYS_INLINE void USB_TransactionCompleteInterruptAck (void)[static]

Clears the USB Transaction Complete Interrupt flag.

Parameters:

None.

Returns:

None.

4.4.14.3.149 USB_TransactionCompleteInterruptDisable()

static ALWAYS_INLINE void USB_TransactionCompleteInterruptDisable (void)[static]

Disables the USB Transaction Complete interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.150 USB_TransactionCompleteInterruptEnable()

static ALWAYS_INLINE void USB_TransactionCompleteInterruptEnable (void)[static]

Enables the USB Transaction Complete interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.151 USB_TransactionCompleteInterruptIs()

static ALWAYS_INLINE bool USB_TransactionCompleteInterruptIs (void)[static]

Checks if a Transaction Complete interrupt has been triggered.

Parameters:

None.

Return values:

0	- Interrupt not triggered
1	- Interrupt triggered

4.4.14.3.152 USB_UnderflowInterruptClear()

static ALWAYS_INLINE void USB_UnderflowInterruptClear (void)[static]

Clears the USB Underflow Interrupt flag.

Parameters:

None.

Returns:

None.

4.4.14.3.153 USB_UnderflowInterruptDisable()

static ALWAYS_INLINE void USB_UnderflowInterruptDisable (void)[static]

Disables the USB Underflow interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.154 USB_UnderflowInterruptEnable()

static ALWAYS_INLINE void USB_UnderflowInterruptEnable (void)[static]

Enables the USB Underflow interrupt.

Parameters:

None.

Returns:

None.

4.4.14.3.155 USB_UnderflowInterruptIs()

static ALWAYS_INLINE bool USB_UnderflowInterruptIs (void)[static]

Checks if an Underflow interrupt has been triggered.

Parameters:

None.

Return values:

0	- Interrupt not triggered
1	- Interrupt triggered

4.4.14.3.156 USB_UpstreamResumeEnable()

static ALWAYS_INLINE void USB_UpstreamResumeEnable (void)[static]

Enables an upstream resume to be initated.

Parameters:

None.

Returns:

None.

4.4.14.3.157 USB_UpstreamResumeIsEnable()

static ALWAYS_INLINE bool USB_UpstreamResumeIsEnable (void)[static]

Checks if upstream resume is enabled, but not yet initiated.

Parameters:

None.

Return values:

0	- Upstream resume initiated or not enabled
1	- Upstream resume enabled

4.4.14.3.158 WaitUntilRMWDone()

static ALWAYS_INLINE void WaitUntilRMWDone (void)[static]

Waits until a Read-Modify-Write operation is done. This blocking wait operation is expected to complete within 14 clock cycles.

Parameters:

None.

Returns:

None.

4.4.15 USB Peripheral Endpoint

API module for usb_peripheral_endpoint covering endpoint-related functions.

4.4.15.1 Module description

API module for usb_peripheral_endpoint covering endpoint-related functions.

Version: USB Device Stack HAL Driver Version 1.0.0

4.4.15.1.1 Definitions

- #define IsPowerOfTwo(number) ((0u != (number)) && (((number) & ((number)-1u)) == 0u))

  Algorithm to detect if a given number is a power of two. A number is a power of two if it has exactly one '1' in its binary representation. This is true if subtracting '1' from the number and doing an AND operation on the result with the number itself returns 0.

4.4.15.1.2 Functions

- RETURN_CODE_t USB_EndpointConfigure (USB_PIPE_t pipe, uint16_t endpointSize, USB_ENDPOINT_t endpointType)

  Configures the endpoint with the desired settings using the Control and Status Register. Used to set up an endpoint before using it in an application. Sets up all the control register settings by looking up the usb_config.h file and clears the count registers.

- RETURN_CODE_t USB_EndpointDisable (USB_PIPE_t pipe)

  Disables the endpoint by setting the endpoint type to 0x00.

- uint16_t USB_EndpointSizeGet (USB_PIPE_t pipe)

  Helper function to return the endpoint size.

- USB_ENDPOINT_t USB_EndpointTypeGet (USB_PIPE_t pipe)

  Helper function to return the endpoint type.

- RETURN_CODE_t USB_EndpointStall (USB_PIPE_t pipe)

  Helps stall an endpoint when a command received from the host is invalid or unrecognizable. Used if the host sends data that is not supported by the device.

- RETURN_CODE_t USB_EndpointStallClear (USB_PIPE_t pipe)

  Helps clear the Stall condition after the device has recovered from an unsupported command from the host. Used to reset stall before the next USB transfer. Used when the host issues a clear HALT/Feature request to reset stall.

- bool USB_EndpointIsStalled (USB_PIPE_t pipe)

  Helper function to return the endpoint Stall condition.

- RETURN_CODE_t USB_EndpointStalledConditionAck (USB_PIPE_t pipe)

  Acknowledges the stall status condition by clearing the Stall Status bit. Used to clear the Stall Status bit after a stall has been detected.

- RETURN_CODE_t USB_DataToggleSet (USB_PIPE_t pipe)

  Sets the Data Toggle bit on an endpoint which is used to ensure correct data sequence. Only used if hardware data toggling is not available. After a successful transaction, toggle the Data Toggle bit. For SETUP transactions, ensure that the SETUP stage clears the Data Toggle bit, while the data stage and status stage set the Data Toggle bit.

- RETURN_CODE_t USB_DataToggleClear (USB_PIPE_t pipe)

  Clears the Data Toggle bit on an endpoint which is used to ensure correct data sequence. Only used if hardware data toggling is not available. After a successful transaction, toggle the Data Toggle bit. For SETUP transactions, ensure that the SETUP stage clears the Data Toggle bit, while the data stage and status stage set the Data Toggle bit.

- RETURN_CODE_t USB_DataToggle (USB_PIPE_t pipe)

  Toggles the Data Toggle bit on an endpoint which is used to ensure correct data sequence. Only used if hardware data toggling is not available. After a successful transaction, toggle the Data

Toggle bit. For SETUP transactions, ensure that the SETUP stage clears the Data Toggle bit, while the data stage and status stage set the Data Toggle bit.

- RETURN_CODE_t ConvertEndpointSizeToMask (uint16_t endpointSize, USB_ENDPOINT_t endpointType, uint8_t *endpointMaskPtr)

  Converts an endpoint size in number of bytes into a register setting. Converts the endpoint size bit mask based on the EP_BUFSIZE setting of the endpoint control register.
- RETURN_CODE_t EndpointBufferSet (USB_PIPE_t pipe, uint8_t *bufAddress)

  Configures the endpoint data buffer to a location in RAM for the next transaction.

4.4.15.1.3 Variables

- USB_ENDPOINT_TABLE_t endpointTable

  SRAM tables for the FIFO and endpoint registers, as well as the FRAMENUM register. Represents the endpoint configuration table based on the number of endpoints in use. This line instantiates an object using the data structure type.

4.4.15.2 Definition Documentation

4.4.15.2.1 IsPowerOfTwo

#define IsPowerOfTwo(number) ((0u != (number)) && (((number) & ((number)-1u)) == 0u))

Algorithm to detect if a given number is a power of two. A number is a power of two if it has exactly one '1' in its binary representation. This is true if subtracting '1' from the number and doing an AND operation on the result with the number itself returns 0.

Parameters:

number	8-bit unsigned integer

Return values:

True	- The given number is a power of two
False	- The given number is not a power of two

4.4.15.3 Function Documentation

4.4.15.3.1 ConvertEndpointSizeToMask()

RETURN_CODE_t ConvertEndpointSizeToMask (uint16_t endpointSize, USB_ENDPOINT_t endpointType, uint8_t * endpointMaskPtr)

Converts an endpoint size in number of bytes into a register setting. Converts the endpoint size bit mask based on the EP_BUFSIZE setting of the endpoint control register.

Parameters:

endpointSize	- The size to convert
endpointType	- The endpoint type
endpointMaskPtr	- Pointer to the mask variable to write to

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.15.3.2 EndpointBufferSet()

RETURN_CODE_t EndpointBufferSet (USB_PIPE_t pipe, uint8_t * bufAddress)

Configures the endpoint data buffer to a location in RAM for the next transaction.

Parameters:

pipe	- A combination of endpoint address and direction
bufAddress	- The pointer to the data buffer the endpoint will use

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.15.3.3 USB_DataToggle()

RETURN_CODE_t USB_DataToggle (USB_PIPE_t pipe)

Toggles the Data Toggle bit on an endpoint which is used to ensure correct data sequence. Only used if hardware data toggling is not available. After a successful transaction, toggle the Data Toggle bit. For SETUP transactions, ensure that the SETUP stage clears the Data Toggle bit, while the data stage and status stage set the Data Toggle bit.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.15.3.4 USB_DataToggleClear()

RETURN_CODE_t USB_DataToggleClear (USB_PIPE_t pipe)

Clears the Data Toggle bit on an endpoint which is used to ensure correct data sequence. Only used if hardware data toggling is not available. After a successful transaction, toggle the Data Toggle bit. For SETUP transactions, ensure that the SETUP stage clears the Data Toggle bit, while the data stage and status stage set the Data Toggle bit.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.15.3.5 USB_DataToggleSet()

RETURN_CODE_t USB_DataToggleSet (USB_PIPE_t pipe)

Sets the Data Toggle bit on an endpoint which is used to ensure correct data sequence. Only used if hardware data toggling is not available. After a successful transaction, toggle the Data Toggle bit. For SETUP transactions, ensure that the SETUP stage clears the Data Toggle bit, while the data stage and status stage set the Data Toggle bit.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.15.3.6 USB_EndpointConfigure()

RETURN_CODE_t USB_EndpointConfigure (USB_PIPE_t pipe, uint16_t endpointSize, USB_ENDPOINT_t endpointType)

Configures the endpoint with the desired settings using the Control and Status Register. Used to set up an endpoint before using it in an application. Sets up all the control register settings by looking up the usb_config.h file and clears the count registers.

Parameters:

pipe	- A combination of endpoint address and direction
endpointSize	- Number of bytes of data supported by the endpoint in one USB transaction
endpointType	- Type of USB endpoint as defined by usb_endpoint_type

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.15.3.7 USB_EndpointDisable()

RETURN_CODE_t USB_EndpointDisable (USB_PIPE_t pipe)

Disables the endpoint by setting the endpoint type to 0x00.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.15.3.8 USB_EndpointIsStalled()

bool USB_EndpointIsStalled (USB_PIPE_t pipe)

Helper function to return the endpoint Stall condition.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

A boolean value representing the Stall condition. If the pipe address is out of bounds, the function will always return false

4.4.15.3.9 USB_EndpointSizeGet()

uint16_t USB_EndpointSizeGet (USB_PIPE_t pipe)

Helper function to return the endpoint size.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

The size of the endpoint

4.4.15.3.10 USB_EndpointStall()

RETURN_CODE_t USB_EndpointStall (USB_PIPE_t pipe)

Helps stall an endpoint when a command received from the host is invalid or unrecognizable. Used if the host sends data that is not supported by the device.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.15.3.11 USB_EndpointStallClear()

RETURN_CODE_t USB_EndpointStallClear (USB_PIPE_t pipe)

Helps clear the Stall condition after the device has recovered from an unsupported command from the host. Used to reset stall before the next USB transfer. Used when the host issues a clear HALT/ Feature request to reset stall.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.15.3.12 USB_EndpointStalledConditionAck()

RETURN_CODE_t USB_EndpointStalledConditionAck (USB_PIPE_t pipe)

Acknowledges the stall status condition by clearing the Stall Status bit. Used to clear the Stall Status bit after a stall has been detected.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.15.3.13 USB_EndpointTypeGet()

USB_ENDPOINT_t USB_EndpointTypeGet (USB_PIPE_t pipe)

Helper function to return the endpoint type.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

The endpoint type

4.4.15.4 Variable Documentation

4.4.15.4.1 endpointTable

USB_ENDPOINT_TABLE_t endpointTable

SRAM tables for the FIFO and endpoint registers, as well as the FRAMENUM register. Represents the endpoint configuration table based on the number of endpoints in use. This line instantiates an object using the data structure type.

4.4.16 USB Peripheral Read/Write

API module for usb_peripheral covering low-level USB transaction functions.

4.4.16.1 Module description

API module for usb_peripheral covering low-level USB transaction functions.

Version: USB Device Stack HAL Driver Version 1.0.0

4.4.16.1.1 Functions

- RETURN_CODE_t USB_TransactionStart (USB_PIPE_t pipe)
  Starts sending or receiving data on an endpoint by clearing BUSNACK. Used as a final step while setting up a transaction on the bus.
- RETURN_CODE_t USB_TransactionAbort (USB_PIPE_t pipe)
  Aborts the next transaction on an endpoint by setting BUSNACK. Used to stop exchanging data on an endpoint. The device will start NAKing requests from the host after calling this API.
- RETURN_CODE_t USB_TransactionCompleteAck (USB_PIPE_t pipe)
  Acknowledges the Transaction Complete status condition by clearing the Transaction Complete status bit. Used to clear the Transaction Complete status bit after a transaction has successfully completed.
- bool USB_TransactionIsCompleted (void)
  Helper function to return the endpoint transaction complete condition.
- RETURN_CODE_t USB_TransactionCompletedPipeGet (USB_PIPE_t *pipe)
  Returns the pipe address and direction for the latest completed transaction.
- RETURN_CODE_t USB_PipeReset (USB_PIPE_t pipe)
  Resets the pipe.
- USB_TRANSFER_STATUS_t USB_PipeStatusGet (USB_PIPE_t pipe)
  Gets the current status of pipe.
- bool USB_PipeStatusIsBusy (USB_PIPE_t pipe)
  Checks if the pipe status is busy.
- void USB_PipeDataPtrSet (USB_PIPE_t pipe, uint8_t *dataPtr)
  Configures the pointer for the data transfer in a given pipe.
- uint8_t * USB_PipeDataPtrGet (USB_PIPE_t pipe)
  Gets the current data pointer for a given pipe.
- void USB_PipeDataToTransferSizeSet (USB_PIPE_t pipe, uint16_t dataSize)
  Sets the size of pipe data to transfer.
- uint16_t USB_PipeDataToTransferSizeGet (USB_PIPE_t pipe)
  Gets the size of pipe data to transfer.
- uint16_t USB_PipeDataTransferredSizeGet (USB_PIPE_t pipe)
  Gets the size of the transferred pipe data.
- void USB_PipeDataTransferredSizeSet (USB_PIPE_t pipe, uint16_t dataSize)
  Sets the size of the transferred pipe data.
- void USB_PipeDataTransferredSizeReset (USB_PIPE_t pipe)
  Resets the size of transferred pipe data.
- void USB_PipeTransferZLP_Enable (USB_PIPE_t pipe)
  Enables a ZLP on a transfer. It is enabled by default if the AZLP static config is enabled for the pipe.
- void USB_PipeTransferEndCallbackRegister (USB_PIPE_t pipe, USB_TRANSFER_END_CALLBACK_t callback)
  Sets the callback for transfer end.
- void USB_PipeTransferEndCallback (USB_PIPE_t pipe)
  Calls the callback for transfer end.
- RETURN_CODE_t USB_InTransactionRun (USB_PIPE_t pipe)

Checks the correctness of IN transactions and runs them.

- RETURN_CODE_t USB_OutTransactionRun (USB_PIPE_t pipe)
  Checks the correctness OUT transactions and runs them.
- RETURN_CODE_t USB_PipeTransactionComplete (USB_PIPE_t pipe)
  Handles completed IN and OUT transactions. Processes the completed transaction and either completes the transfer or runs the next transaction. Will call the pipe transferEndCallback at the end of transfer, if configured.

4.4.16.2 Function Documentation

4.4.16.2.1 USB_InTransactionRun()

RETURN_CODE_t USB_InTransactionRun (USB_PIPE_t pipe)

Checks the correctness of IN transactions and runs them.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.16.2.2 USB_OutTransactionRun()

RETURN_CODE_t USB_OutTransactionRun (USB_PIPE_t pipe)

Checks the correctness OUT transactions and runs them.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.16.2.3 USB_PipeDataPtrGet()

uint8_t* USB_PipeDataPtrGet (USB_PIPE_t pipe)

Gets the current data pointer for a given pipe.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

The pointer to the data location

4.4.16.2.4 USB_PipeDataPtrSet()

void USB_PipeDataPtrSet (USB_PIPE_t pipe, uint8_t * dataPtr)

Configures the pointer for the data transfer in a given pipe.

Parameters:

pipe	- A combination of endpoint address and direction
*dataPtr	- The pointer to the data location

Returns:

None.

4.4.16.2.5 USB_PipeDataToTransferSizeGet()

uint16_t USB_PipeDataToTransferSizeGet (USB_PIPE_t pipe)

Gets the size of pipe data to transfer.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

The size of pipe data to transfer

4.4.16.2.6 USB_PipeDataToTransferSizeSet()

void USB_PipeDataToTransferSizeSet (USB_PIPE_t pipe, uint16_t dataSize)

Sets the size of pipe data to transfer.

Parameters:

pipe	- A combination of endpoint address and direction
dataSize	- The size of pipe data to transfer

Returns:

None.

4.4.16.2.7 USB_PipeDataTransferredSizeGet()

uint16_t USB_PipeDataTransferredSizeGet (USB_PIPE_t pipe)

Gets the size of the transferred pipe data.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

The size of transferred pipe data

4.4.16.2.8 USB_PipeDataTransferredSizeReset()

void USB_PipeDataTransferredSizeReset (USB_PIPE_t pipe)

Resets the size of transferred pipe data.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

None.

4.4.16.2.9 USB_PipeDataTransferredSizeSet()

void USB_PipeDataTransferredSizeSet (USB_PIPE_t pipe, uint16_t dataSize)

Sets the size of the transferred pipe data.

Parameters:

pipe	- A combination of endpoint address and direction
dataSize	- The size of pipe data transferred

Returns:

None.

4.4.16.2.10 USB_PipeReset()

RETURN_CODE_t USB_PipeReset (USB_PIPE_t pipe)

Resets the pipe.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.16.2.11 USB_PipeStatusGet()

USB_TRANSFER_STATUS_t USB_PipeStatusGet (USB_PIPE_t pipe)

Gets the current status of pipe.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

USB_PIPE_TRANSFER_OK or an Error code according to USB_TRANSFER_STATUS_t

4.4.16.2.12 USB_PipeStatusIsBusy()

bool USB_PipeStatusIsBusy (USB_PIPE_t pipe)

Checks if the pipe status is busy.

Parameters:

pipe	- A combination of endpoint address and direction

Return values:

0	- Pipe status not busy
1	- Pipe status is busy

4.4.16.2.13 USB_PipeTransactionComplete()

RETURN_CODE_t USB_PipeTransactionComplete (USB_PIPE_t pipe)

Handles completed IN and OUT transactions. Processes the completed transaction and either completes the transfer or runs the next transaction. Will call the pipe transferEndCallback at the end of transfer, if configured.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.16.2.14 USB_PipeTransferEndCallback()

void USB_PipeTransferEndCallback (USB_PIPE_t pipe)

Calls the callback for transfer end.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

None.

4.4.16.2.15 USB_PipeTransferEndCallbackRegister()

void USB_PipeTransferEndCallbackRegister (USB_PIPE_t pipe, USB_TRANSFER_END_CALLBACK_t callback)

Sets the callback for transfer end.

Parameters:

pipe	- A combination of endpoint address and direction
callback	- A combination of pipe, status and transferred bytes

Returns:

None.

4.4.16.2.16 USB_PipeTransferZLP_Enable()

void USB_PipeTransferZLP_Enable (USB_PIPE_t pipe)

Enables a ZLP on a transfer. It is enabled by default if the AZLP static config is enabled for the pipe.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

None.

4.4.16.2.17 USB_TransactionAbort()

RETURN_CODE_t USB_TransactionAbort (USB_PIPE_t pipe)

Aborts the next transaction on an endpoint by setting BUSNACK. Used to stop exchanging data on an endpoint. The device will start NAKing requests from the host after calling this API.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.16.2.18 USB_TransactionCompleteAck()

RETURN_CODE_t USB_TransactionCompleteAck (USB_PIPE_t pipe)

Acknowledges the Transaction Complete status condition by clearing the Transaction Complete status bit. Used to clear the Transaction Complete status bit after a transaction has successfully completed.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.16.2.19 USB_TransactionCompletedPipeGet()

RETURN_CODE_t USB_TransactionCompletedPipeGet (USB_PIPE_t * pipe)

Returns the pipe address and direction for the latest completed transaction.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.16.2.20 USB_TransactionIsCompleted()

bool USB_TransactionIsCompleted (void)

Helper function to return the endpoint transaction complete condition.

Parameters:

None.

Return values:

0	- Transaction not complete or pipe address is out of bounds
1	- Transaction is complete

4.4.16.2.21 USB_TransactionStart()

RETURN_CODE_t USB_TransactionStart (USB_PIPE_t pipe)

Starts sending or receiving data on an endpoint by clearing BUSNACK. Used as a final step while setting up a transaction on the bus.

Parameters:

pipe	- A combination of endpoint address and direction

Returns:

SUCCESS or an Error code according to RETURN_CODE_t

4.4.17 USB Vendor Class

Contains the prototypes and data types for the vendor application drivers.

4.4.17.1 Module description

Contains the prototypes and data types for the vendor application drivers.

Version: USB Device Stack Driver Version 1.0.0

4.4.17.1.1 Functions

- void USB_VendorClassInit (USB_SETUP_EVENT_CALLBACK_t interfaceEnabledCallback, USB_SETUP_PROCESS_CALLBACK_t vendorControlRequest, USB_EVENT_CALLBACK_t interfaceDisabledCallback)

  Sets up interfaces for use with the Vendor class.

4.4.17.2 Function Documentation

4.4.17.2.1 USB_VendorClassInit()

void USB_VendorClassInit (USB_SETUP_EVENT_CALLBACK_t interfaceEnabledCallback, USB_SETUP_PROCESS_CALLBACK_t vendorControlRequest, USB_EVENT_CALLBACK_t interfaceDisabledCallback)

Sets up interfaces for use with the Vendor class.

Parameters:

interfaceEnabledCallback	- Callback to the interface enable function
vendorControlRequest	- Callback to the control request function
interfaceDisabledCallback	- Callback to the interface disable function

Returns:

None.

4.5 Data Structure Documentation

4.5.1 CIRCULAR_BUFFER_struct Struct Reference

#include <circular_buffer.h>

4.5.1.1 Data Fields

- uint8_t * content
- uint16_t head
- uint16_t tail
- const uint16_t maxLength

4.5.1.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/circular_buffer/

circular_buffer.h

4.5.1.2.1 content

uint8_t* content

Actual buffer data

4.5.1.2.2 head

uint16_t head

Index of first empty slot in buffer

4.5.1.2.3 maxLength

const uint16_t maxLength

Maximum length of buffer

4.5.1.2.4 tail

uint16_t tail

Index of first occupied buffer slot

4.5.2 CIRCULAR_BUFFER_t Struct Reference

Type define for circular buffers of varying length.

4.5.2.1 Detailed Description

Type define for circular buffers of varying length.

#include <circular_buffer.h>

The documentation for this struct was generated from the following file:

source/source-files/circular_buffer/

circular_buffer.h

4.5.3 USB_ASSOCIATION_DESC_struct Struct Reference

#include <usb_protocol_headers.h>

4.5.3.1 Data Fields

- USB_DESCRIPTOR_HEADER_t header
- uint8_t bFirstInterface
- uint8_t bInterfaceCount
- uint8_t bFunctionClass
- uint8_t bFunctionSubClass
- uint8_t bFunctionProtocol
- uint8_t iFunction

4.5.3.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.3.2.1 bFirstInterface

uint8_t bFirstInterface

Number of interface

4.5.3.2.2 bFunctionClass

uint8_t bFunctionClass

Class code assigned by the USB

4.5.3.2.3 bFunctionProtocol

uint8_t bFunctionProtocol

Protocol code assigned by the USB

4.5.3.2.4 bFunctionSubClass

uint8_t bFunctionSubClass

Sub-class code assigned by the USB

4.5.3.2.5 bInterfaceCount

uint8_t bInterfaceCount

Value to select alternate setting

4.5.3.2.6 header

USB_DESCRIPTOR_HEADER_t header

Descriptor type and size

4.5.3.2.7 iFunction

uint8_t iFunction

Index of string descriptor

4.5.4 USB_CDC_ACM_FUNCTIONAL_DESCRIPTOR_struct Struct Reference

#include <usb_protocol_cdc.h>

4.5.4.1 Data Fields

- uint8_t bLength
- uint8_t bDescriptorType
- uint8_t bDescriptorSubtype
- uint8_t bmCapabilities

4.5.4.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/

usb_protocol_cdc.h

4.5.4.2.1 bDescriptorSubtype

uint8_t bDescriptorSubtype

CDC functional descriptor subtype

4.5.4.2.2 bDescriptorType

uint8_t bDescriptorType

CDC functional descriptor type

4.5.4.2.3 bLength

uint8_t bLength

Size of this descriptor in bytes

4.5.4.2.4 bmCapabilities

uint8_t bmCapabilities

The capabilities that this configuration supports. A bit value of zero means that the request is not supported.

4.5.5 USB_CDC_ACM_FUNCTIONAL_DESCRIPTOR_t Struct Reference

Type define for the CDC Abstract Control Management functional descriptor subtype.

4.5.5.1 Detailed Description

Type define for the CDC Abstract Control Management functional descriptor subtype.

#include <usb_protocol_cdc.h>

The documentation for this struct was generated from the following file:

source/source-files/

usb_protocol_cdc.h

4.5.6 USB_CDC_COUNTRY_SELECTION_FUNCTIONAL_DESCRIPTOR_struct Struct Reference

#include <usb_protocol_cdc.h>

4.5.6.1 Data Fields

- uint8_t bLength
- uint8_t bDescriptorType
- uint8_t bDescriptorSubtype
- uint8_t iCountryCodeRelDate
- uint16_t wCountryCode0

4.5.6.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/

usb_protocol_cdc.h

4.5.6.2.1 bDescriptorSubtype

uint8_t bDescriptorSubtype

CDC functional descriptor subtype

4.5.6.2.2 bDescriptorType

uint8_t bDescriptorType

CDC functional descriptor type

4.5.6.2.3 bLength

uint8_t bLength

Size of this descriptor in bytes

4.5.6.2.4 iCountryCodeRelDate

uint8_t iCountryCodeRelDate

Index of the release date in ISO 3166 Country Codes

4.5.6.2.5 wCountryCode0

uint16_t wCountryCode0

First country code

4.5.7 USB_CDC_COUNTRY_SELECTION_FUNCTIONAL_DESCRIPTOR_t Struct Reference

Type define for the CDC country selection functional descriptor subtype.

4.5.7.1 Detailed Description

Type define for the CDC country selection functional descriptor subtype.

#include <usb_protocol_cdc.h>

The documentation for this struct was generated from the following file:

source/source-files/

usb_protocol_cdc.h

4.5.8 USB_CDC_HEADER_FUNCTIONAL_DESCRIPTOR_struct Struct Reference

#include <usb_protocol_cdc.h>

4.5.8.1 Data Fields

- uint8_t bLength
- uint8_t bDescriptorType
- uint8_t bDescriptorSubtype
- uint16_t bcdCDC

4.5.8.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/

usb_protocol_cdc.h

4.5.8.2.1 bcdCDC

uint16_t bcdCDC

USB Class Definitions for Communications Devices Specification release number in binary-coded decimal

4.5.8.2.2 bDescriptorSubtype

uint8_t bDescriptorSubtype

CDC functional descriptor subtype

4.5.8.2.3 bDescriptorType

uint8_t bDescriptorType

CDC functional descriptor type

4.5.8.2.4 bLength

uint8_t bLength

Size of this descriptor in bytes

4.5.9 USB_CDC_HEADER_FUNCTIONAL_DESCRIPTOR_t Struct Reference

Type define for the CDC Header functional descriptor subtype.

4.5.9.1 Detailed Description

Type define for the CDC Header functional descriptor subtype.

#include <usb_protocol_cdc.h>

The documentation for this struct was generated from the following file:

source/source-files/

usb_protocol_cdc.h

4.5.10 USB_CDC_LINE_CODING_struct Struct Reference

#include <usb_protocol_cdc.h>

4.5.10.1 Data Fields

- uint32_t dwDTERate
- USB_CDC_LINE_CODING_STOP_BITS_t bCharFormat
- USD_CDC_LINE_CODING_PARITY_t bParityType
- USD_CDC_LINE_CODING_DATA_BITS_t bDataBits

4.5.10.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/

usb_protocol_cdc.h

4.5.10.2.1 bCharFormat

USB_CDC_LINE_CODING_STOP_BITS_t bCharFormat

Number of stop bits

4.5.10.2.2 bDataBits

USD_CDC_LINE_CODING_DATA_BITS_t bDataBits

Number of data bits

4.5.10.2.3 bParityType

USD_CDC_LINE_CODING_PARITY_t bParityType

Parity control mode

4.5.10.2.4 dwDTERate

uint32_t dwDTERate

Data terminal rate, in bits per second

4.5.11 USB_CDC_LINE_CODING_t Struct Reference

Type define for CDC Line Encoding.

4.5.11.1 Detailed Description

Type define for CDC Line Encoding.

#include <usb_protocol_cdc.h>

The documentation for this struct was generated from the following file:

source/source-files/

usb_protocol_cdc.h

4.5.12 USB_CONFIGURATION_DESCRIPTOR_struct Struct Reference

#include <usb_protocol_headers.h>

4.5.12.1 Data Fields

- USB_DESCRIPTOR_HEADER_t header
- uint16_t wTotalLength
- uint8_t bNumInterfaces
- uint8_t bConfigurationValue
- uint8_t iConfiguration
- uint8_t bmAttributes
- uint8_t bMaxPower

4.5.12.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.12.2.1 bConfigurationValue

uint8_t bConfigurationValue

4.5.12.2.2 bmAttributes

uint8_t bmAttributes

4.5.12.2.3 bMaxPower

uint8_t bMaxPower

4.5.12.2.4 bNumInterfaces

uint8_t bNumInterfaces

4.5.12.2.5 header

USB_DESCRIPTOR_HEADER_t header

Descriptor type and size

4.5.12.2.6 iConfiguration

uint8_t iConfiguration

4.5.12.2.7 wTotalLength

uint16_t wTotalLength

4.5.13 USB_CONTROL_TRANSFER_struct Struct Reference

#include <usb_peripheral.h>

4.5.13.1 Data Fields

- uint8_t buffer [64]
- volatile USB_CONTROL_STATUS_t status
- uint8_t * transferDataPtr
- uint16_t transferDataSize
- uint16_t totalBytesTransferred
- USB_SETUP_PROCESS_CALLBACK_t processSetupCallback
- USB_SETUP_OVERUNDERRUN_CALLBACK_t overUnderRunCallback
- USB_SETUP_ENDOFREQUEST_CALLBACK_t endOfRequestCallback
- USB_SETUP_REQUEST_t setupRequest

4.5.13.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_peripheral/

usb_peripheral.h

4.5.13.2.1 buffer

uint8_t buffer[64]

Default buffer for control data transfers

4.5.13.2.2 endOfRequestCallback

USB_SETUP_ENDOFREQUEST_CALLBACK_t endOfRequestCallback

Callback to call when a setup request is complete

4.5.13.2.3 overUnderRunCallback

USB_SETUP_OVERUNDERRUN_CALLBACK_t overUnderRunCallback

Callback to call on a control overrun or underrun

4.5.13.2.4 processSetupCallback

USB_SETUP_PROCESS_CALLBACK_t processSetupCallback

Callback to call during setup process

4.5.13.2.5 setupRequest

USB_SETUP_REQUEST_t setupRequest

Setup request packet

4.5.13.2.6 status

volatile USB_CONTROL_STATUS_t status

The status of a transfer on this pipe

4.5.13.2.7 totalBytesTransferred

uint16_t totalBytesTransferred

Number of data transfered last transaction

4.5.13.2.8 transferDataPtr

uint8_t* transferDataPtr

Location in RAM to send or fill during transfer

4.5.13.2.9 transferDataSize

uint16_t transferDataSize

Number of bytes to transfer to or from RAM location

4.5.14 USB_CONTROL_TRANSFER_t Struct Reference

The data structure for internally handling control transfers, either IN or OUT.

4.5.14.1 Detailed Description

The data structure for internally handling control transfers, either IN or OUT.

#include <usb_peripheral.h>

The documentation for this struct was generated from the following file:

source/source-files/usb_peripheral/

usb_peripheral.h

4.5.15 USB_DESCRIPTOR_HEADER_struct Struct Reference

#include <usb_protocol_headers.h>

4.5.15.1 Data Fields

- uint8_t bLength
- uint8_t bDescriptorType

4.5.15.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.15.2.1 bDescriptorType

uint8_t bDescriptorType

Descriptor type constant - USB_DESCRIPTOR_TYPE_DEVICE

4.5.15.2.2 bLength

uint8_t bLength

Descriptor size in bytes - sizeof(USB_DEVICE_DESCRIPTOR_t)

4.5.16 USB_DESCRIPTOR_POINTERS_struct Struct Reference

#include <usb_protocol_headers.h>

4.5.16.1 Data Fields

- USB_DEVICE_DESCRIPTOR_t * devicePtr
- USB_CONFIGURATION_DESCRIPTOR_t * configurationsPtr
- USB_DEV_BOS_DESC_t * deviceBOSptr
- USB_STRING_LANG_ID_DESCRIPTOR_t * langIDptr
- USB_DESCRIPTOR_HEADER_t * stringPtrs [LANG_ID_NUM]

4.5.16.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.16.2.1 configurationsPtr
USB_CONFIGURATION_DESCRIPTOR_t* configurationsPtr

4.5.16.2.2 deviceBOSptr
USB_DEV_BOS_DESC_t* deviceBOSptr

4.5.16.2.3 devicePtr
USB_DEVICE_DESCRIPTOR_t* devicePtr

4.5.16.2.4 langIDptr
USB_STRING_LANG_ID_DESCRIPTOR_t* langIDptr

4.5.16.2.5 stringPtrs
USB_DESCRIPTOR_HEADER_t* stringPtrs[LANG_ID_NUM]

4.5.17 USB_DESCRIPTOR_PTR_union Union Reference

4.5.17.1 Data Fields
- uint8_t * bytePtr
- USB_DESCRIPTOR_HEADER_t * headerPtr
- USB_ENDPOINT_DESCRIPTOR_t * endpointPtr
- USB_INTERFACE_DESCRIPTOR_t * interfacePtr
- USB_CONFIGURATION_DESCRIPTOR_t * configurationPtr

4.5.17.2 Field Documentation
The documentation for this union was generated from the following file:

source/source-files/usb_common/

usb_core_descriptors.c

4.5.17.2.1 bytePtr
uint8_t* bytePtr

4.5.17.2.2 configurationPtr
USB_CONFIGURATION_DESCRIPTOR_t* configurationPtr

4.5.17.2.3 endpointPtr
USB_ENDPOINT_DESCRIPTOR_t* endpointPtr

4.5.17.2.4 headerPtr
USB_DESCRIPTOR_HEADER_t* headerPtr

4.5.17.2.5 interfacePtr
USB_INTERFACE_DESCRIPTOR_t* interfacePtr

4.5.18 USB_DEV_BOS_DESC_struct Struct Reference
#include <usb_protocol_headers.h>

4.5.18.1 Data Fields
- USB_DESCRIPTOR_HEADER_t header
- uint16_t wTotalLength
- uint8_t bNumDeviceCaps

4.5.18.2 Field Documentation
The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.18.2.1 bNumDeviceCaps

uint8_t bNumDeviceCaps

4.5.18.2.2 header

USB_DESCRIPTOR_HEADER_t header

Descriptor type and size

4.5.18.2.3 wTotalLength

uint16_t wTotalLength

4.5.19 USB_DEV_CAPA_EXT_DESC_struct Struct Reference

#include <usb_protocol_headers.h>

4.5.19.1 Data Fields

- USB_DESCRIPTOR_HEADER_t header
- uint8_t bDevCapabilityType
- uint32_t bmAttributes

4.5.19.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.19.2.1 bDevCapabilityType

uint8_t bDevCapabilityType

4.5.19.2.2 bmAttributes

uint32_t bmAttributes

4.5.19.2.3 header

USB_DESCRIPTOR_HEADER_t header

Descriptor type and size

4.5.20 USB_DEV_LPM_DESC_struct Struct Reference

#include <usb_protocol_headers.h>

4.5.20.1 Data Fields

- USB_DEV_BOS_DESC_t bos
- USB_DEV_CAPA_EXT_DESC_t capa_ext

4.5.20.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.20.2.1 bos

USB_DEV_BOS_DESC_t bos

4.5.20.2.2 capa_ext

USB_DEV_CAPA_EXT_DESC_t capa_ext

4.5.21 USB_DEV_QUAL_DESC_struct Struct Reference

#include <usb_protocol_headers.h>

4.5.21.1 Data Fields

- USB_DESCRIPTOR_HEADER_t header
- uint16_t bcdUSB
- uint8_t bDeviceClass
- uint8_t bDeviceSubClass
- uint8_t bDeviceProtocol
- uint8_t bMaxPacketSize0
- uint8_t bNumConfigurations
- uint8_t bReserved

4.5.21.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.21.2.1 bcdUSB

uint16_t bcdUSB

4.5.21.2.2 bDeviceClass

uint8_t bDeviceClass

4.5.21.2.3 bDeviceProtocol

uint8_t bDeviceProtocol

4.5.21.2.4 bDeviceSubClass

uint8_t bDeviceSubClass

4.5.21.2.5 bMaxPacketSize0

uint8_t bMaxPacketSize0

4.5.21.2.6 bNumConfigurations

uint8_t bNumConfigurations

4.5.21.2.7 bReserved

uint8_t bReserved

4.5.21.2.8 header

USB_DESCRIPTOR_HEADER_t header

Descriptor type and size

4.5.22 USB_DEVICE_DESCRIPTOR_struct Struct Reference

#include <usb_protocol_headers.h>

4.5.22.1 Data Fields

- USB_DESCRIPTOR_HEADER_t header
- uint16_t bcdUSB
- uint8_t bDeviceClass
- uint8_t bDeviceSubClass
- uint8_t bDeviceProtocol

- uint8_t bMaxPacketSize0
- uint16_t idVendor
- uint16_t idProduct
- uint16_t bcdDevice
- uint8_t iManufacturer
- uint8_t iProduct
- uint8_t iSerialNumber
- uint8_t bNumConfigurations

4.5.22.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.22.2.1 bcdDevice

uint16_t bcdDevice

Device release number in BCD format

4.5.22.2.2 bcdUSB

uint16_t bcdUSB

USB spec release number in BCD format

4.5.22.2.3 bDeviceClass

uint8_t bDeviceClass

Device Class Code

4.5.22.2.4 bDeviceProtocol

uint8_t bDeviceProtocol

Device Protocol Code

4.5.22.2.5 bDeviceSubClass

uint8_t bDeviceSubClass

Device SubClass Code

4.5.22.2.6 bMaxPacketSize0

uint8_t bMaxPacketSize0

Maximum packet size for endpoint 0

4.5.22.2.7 bNumConfigurations

uint8_t bNumConfigurations

Number of possible configurations

4.5.22.2.8 header

USB_DESCRIPTOR_HEADER_t header

Descriptor type and size

4.5.22.2.9 idProduct

uint16_t idProduct

Product ID

4.5.22.2.10 idVendor

uint16_t idVendor

Vendor ID

4.5.22.2.11 iManufacturer

uint8_t iManufacturer

Manufacturer string descriptor index

4.5.22.2.12 iProduct

uint8_t iProduct

Product string descriptor index

4.5.22.2.13 iSerialNumber

uint8_t iSerialNumber

Device serial number string descriptor index

4.5.23 USB_ENDPOINT_DESCRIPTOR_struct Struct Reference

#include <usb_protocol_headers.h>

4.5.23.1 Data Fields

- USB_DESCRIPTOR_HEADER_t header
- USB_PIPE_t bEndpointAddress
- struct {
  - uint8_t type: 2
  - uint8_t synchronisation: 2
  - uint8_t usage: 2
  - uint8_t reserved: 2
- } bmAttributes
- uint16_t wMaxPacketSize
- uint8_t bInterval

4.5.23.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.23.2.1 bEndpointAddress

USB_PIPE_t bEndpointAddress

4.5.23.2.2 bInterval

uint8_t bInterval

4.5.23.2.3 bmAttributes

struct { ... } bmAttributes

4.5.23.2.4 header

USB_DESCRIPTOR_HEADER_t header

Descriptor type and size

4.5.23.2.5 reserved

uint8_t reserved

4.5.23.2.6 synchronisation

uint8_t synchronisation

4.5.23.2.7 type

uint8_t type

4.5.23.2.8 usage

uint8_t usage

4.5.23.2.9 wMaxPacketSize

uint16_t wMaxPacketSize

4.5.24 USB_ENDPOINT_TABLE_struct Struct Reference

Represents the endpoint configuration table based on the number of endpoints in use. The table data structure is defined by USB_EP_TABLE_struct in the device header file, modified to support configuration of size from USB_EP_NUM.

4.5.24.1 Detailed Description

Represents the endpoint configuration table based on the number of endpoints in use. The table data structure is defined by USB_EP_TABLE_struct in the device header file, modified to support configuration of size from USB_EP_NUM.

#include <usb_peripheral_avr_du.h>

4.5.24.1.1 Data Fields

- register8_t FIFO [USB_EP_NUM *2u]
- USB_EP_PAIR_t EP [USB_EP_NUM]
- register16_t FRAMENUM

4.5.24.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_peripheral/

usb_peripheral_avr_du.h

4.5.24.2.1 EP

USB_EP_PAIR_t EP[USB_EP_NUM]

USB Endpoint Register Pairs

4.5.24.2.2 FIFO

register8_t FIFO[USB_EP_NUM *2u]

FIFO Entry

4.5.24.2.3 FRAMENUM

register16_t FRAMENUM

Frame Number

4.5.25 USB_EVENT_HANDLERS_struct Struct Reference

Represents the event callbacks, the configuration and the enumeration setups.

4.5.25.1 Detailed Description

Represents the event callbacks, the configuration and the enumeration setups.

#include <usb_core_events.h>

4.5.25.1.1 Data Fields

- USB_SETUP_EVENT_CALLBACK_t SetConfiguration
- USB_SETUP_EVENT_CALLBACK_t SetInterface
- USB_EVENT_CALLBACK_t InterfaceDisabled
- USB_SETUP_PROCESS_CALLBACK_t VendorRequest
- USB_SETUP_PROCESS_CALLBACK_t ClassRequest
- USB_SETUP_PROCESS_CALLBACK_t OtherRequest
- USB_EVENT_CALLBACK_t SOFCallback
- USB_EVENT_CALLBACK_t ResetCallback
- USB_EVENT_CALLBACK_t SuspendCallback
- USB_EVENT_CALLBACK_t ResumeCallback

4.5.25.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_core_events.h

4.5.25.2.1 ClassRequest

USB_SETUP_PROCESS_CALLBACK_t ClassRequest

4.5.25.2.2 InterfaceDisabled

USB_EVENT_CALLBACK_t InterfaceDisabled

4.5.25.2.3 OtherRequest

USB_SETUP_PROCESS_CALLBACK_t OtherRequest

4.5.25.2.4 ResetCallback

USB_EVENT_CALLBACK_t ResetCallback

4.5.25.2.5 ResumeCallback

USB_EVENT_CALLBACK_t ResumeCallback

4.5.25.2.6 SetConfiguration

USB_SETUP_EVENT_CALLBACK_t SetConfiguration

4.5.25.2.7 SetInterface

USB_SETUP_EVENT_CALLBACK_t SetInterface

4.5.25.2.8 SOFCallback

USB_EVENT_CALLBACK_t SOFCallback

4.5.25.2.9 SuspendCallback

USB_EVENT_CALLBACK_t SuspendCallback

4.5.25.2.10 VendorRequest

USB_SETUP_PROCESS_CALLBACK_t VendorRequest

4.5.26 USB_HID_DESCRIPTOR_t Struct Reference

Type defines for a standard HID descriptor.

4.5.26.1 Detailed Description

Type defines for a standard HID descriptor.

#include <usb_protocol_hid.h>

4.5.26.1.1 Data Fields

- USB_DESCRIPTOR_HEADER_t header
- uint16_t bcdHID
- uint8_t bCountryCode
- uint8_t bNumDescriptors
- uint8_t bRDescriptorType
- uint16_t wDescriptorLength

4.5.26.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/

usb_protocol_hid.h

4.5.26.2.1 bcdHID

uint16_t bcdHID

Binary Coded Decimal Spec. release

4.5.26.2.2 bCountryCode

uint8_t bCountryCode

Hardware target country

4.5.26.2.3 bNumDescriptors

uint8_t bNumDescriptors

Number of HID class descriptors to follow

4.5.26.2.4 bRDescriptorType

uint8_t bRDescriptorType

Report descriptor type

4.5.26.2.5 header

USB_DESCRIPTOR_HEADER_t header

Descriptor type and size

4.5.26.2.6 wDescriptorLength

uint16_t wDescriptorLength

Total length of the Report descriptor

4.5.27 USB_HID_REPORT_DESCRIPTOR_t Struct Reference

Report descriptor for a HID application.

4.5.27.1 Detailed Description

Report descriptor for a HID application.

#include <usb_protocol_hid.h>

4.5.27.1.1 Data Fields

- uint8_t array [USB_HID_REPORT_DESCRIPTOR_SIZE]

4.5.27.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/

usb_protocol_hid.h

4.5.27.2.1 array
uint8_t array[USB_HID_REPORT_DESCRIPTOR_SIZE]

4.5.28 USB_IAD_DESC_struct Struct Reference
Standard USB association descriptor structure.

4.5.28.1 Detailed Description
Standard USB association descriptor structure.

#include <usb_protocol_headers.h>

4.5.28.1.1 Data Fields
- USB_DESCRIPTOR_HEADER_t header
- uint8_t bFirstInterface
- uint8_t bInterfaceCount
- uint8_t bFunctionClass
- uint8_t bFunctionSubClass
- uint8_t bFunctionProtocol
- uint8_t iFunction

4.5.28.2 Field Documentation
The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.28.2.1 bFirstInterface
uint8_t bFirstInterface

Number of interface

4.5.28.2.2 bFunctionClass
uint8_t bFunctionClass

Class code assigned by the USB

4.5.28.2.3 bFunctionProtocol
uint8_t bFunctionProtocol

Protocol code assigned by the USB

4.5.28.2.4 bFunctionSubClass
uint8_t bFunctionSubClass

Sub-class code assigned by the USB

4.5.28.2.5 bInterfaceCount
uint8_t bInterfaceCount

Value to select alternate setting

4.5.28.2.6 header
USB_DESCRIPTOR_HEADER_t header

Descriptor type and size

4.5.28.2.7 iFunction
uint8_t iFunction

Index of string descriptor

4.5.29 USB_INTERFACE_DESCRIPTOR_struct Struct Reference

Ch. 9.6.5 Standard USB interface descriptor structure.

4.5.29.1 Detailed Description

Ch. 9.6.5 Standard USB interface descriptor structure.

#include <usb_protocol_headers.h>

4.5.29.1.1 Data Fields

- USB_DESCRIPTOR_HEADER_t header
- uint8_t bInterfaceNumber
- uint8_t bAlternateSetting
- uint8_t bNumEndpoints
- uint8_t bInterfaceClass
- uint8_t bInterfaceSubClass
- uint8_t bInterfaceProtocol
- uint8_t iInterface

4.5.29.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.29.2.1 bAlternateSetting

uint8_t bAlternateSetting

4.5.29.2.2 bInterfaceClass

uint8_t bInterfaceClass

4.5.29.2.3 bInterfaceNumber

uint8_t bInterfaceNumber

4.5.29.2.4 bInterfaceProtocol

uint8_t bInterfaceProtocol

4.5.29.2.5 bInterfaceSubClass

uint8_t bInterfaceSubClass

4.5.29.2.6 bNumEndpoints

uint8_t bNumEndpoints

4.5.29.2.7 header

USB_DESCRIPTOR_HEADER_t header

Descriptor type and size

4.5.29.2.8 iInterface

uint8_t iInterface

4.5.30 USB_KEYBOARD_REPORT_DATA_t Struct Reference

Type defines for a standard keyboard input report.

4.5.30.1 Detailed Description

Type defines for a standard keyboard input report.

#include <usb_protocol_hid.h>

4.5.30.1.1 Data Fields

- uint8_t Modifier
- uint8_t Reserved
- uint8_t KeyCode [6]

4.5.30.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/

usb_protocol_hid.h

4.5.30.2.1 KeyCode

uint8_t KeyCode[6]

Key codes of the currently pressed keys

4.5.30.2.2 Modifier

uint8_t Modifier

Keyboard modifier byte

4.5.30.2.3 Reserved

uint8_t Reserved

Reserved for OEM use, always set to 0

4.5.31 USB_MOUSE_REPORT_DATA_t Struct Reference

Type defines for a standard mouse input report.

4.5.31.1 Detailed Description

Type defines for a standard mouse input report.

#include <usb_protocol_hid.h>

4.5.31.1.1 Data Fields

- uint8_t Button
- int8_t X
- int8_t Y

4.5.31.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/

usb_protocol_hid.h

4.5.31.2.1 Button

uint8_t Button

Button mask for currently pressed buttons in the mouse

4.5.31.2.2 X

int8_t X

Current delta X movement of the mouse

4.5.31.2.3 Y

int8_t Y

Current delta Y movement of the mouse

4.5.32 USB_PIPE_struct Struct Reference

Structure for a pipe which has an address and direction.

4.5.32.1 Detailed Description

Structure for a pipe which has an address and direction.

#include <usb_protocol_headers.h>

4.5.32.1.1 Data Fields

- uint8_t address: 7
- uint8_t direction: 1

4.5.32.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.32.2.1 address

uint8_t address

4.5.32.2.2 direction

uint8_t direction

4.5.33 USB_PIPE_TRANSFER_struct Struct Reference

Represents a transfer created for a pipe, either IN or OUT.

4.5.33.1 Detailed Description

Represents a transfer created for a pipe, either IN or OUT.

#include <usb_peripheral_avr_du.h>

4.5.33.1.1 Data Fields

- USB_TRANSFER_STATUS_t status
- uint8_t ZLPEnable: 1
- uint8_t reserved: 7
- uint8_t * transferDataPtr
- uint16_t transferDataSize
- uint16_t bytesTransferred
- USB_TRANSFER_END_CALLBACK_t transferEndCallback

4.5.33.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_peripheral/

usb_peripheral_avr_du.h

4.5.33.2.1 bytesTransferred

uint16_t bytesTransferred

Total number of data transfered

4.5.33.2.2 reserved

uint8_t reserved

These bits are unused

4.5.33.2.3 status

USB_TRANSFER_STATUS_t status

The status of a transfer on this pipe

4.5.33.2.4 transferDataPtr

uint8_t* transferDataPtr

Location in RAM to send or fill during transfer

4.5.33.2.5 transferDataSize

uint16_t transferDataSize

Number of bytes to transfer to or from the RAM location

4.5.33.2.6 transferEndCallback

USB_TRANSFER_END_CALLBACK_t transferEndCallback

Callback to call at the end of transfer when transfer_data_size == bytes_transfered, NULL if not used

4.5.33.2.7 ZLPEnable

uint8_t ZLPEnable

A Zero Length Packet (ZLP) is enabled for the end of this transfer if the transfer size is a multiple of endpoint size

4.5.34 USB_SETUP_REQUEST_struct Struct Reference

#include <usb_protocol_headers.h>

4.5.34.1 Data Fields

- struct {
  - uint8_t recipient: 5
  - uint8_t type: 2
  - uint8_t dataPhaseTransferDirection: 1
- } bmRequestType
- uint8_t bRequest
- uint16_t wValue
- uint16_t wIndex
- uint16_t wLength

4.5.34.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.34.2.1 bmRequestType

struct { ... } bmRequestType

4.5.34.2.2 bRequest

uint8_t bRequest

Request identifier

4.5.34.2.3 dataPhaseTransferDirection

uint8_t dataPhaseTransferDirection

Data stage direction bit field

4.5.34.2.4 recipient

uint8_t recipient

Recipient type bit field

4.5.34.2.5 type

uint8_t type

Request type bit field

4.5.34.2.6 wIndex

uint16_t wIndex

Request specific index

4.5.34.2.7 wLength

uint16_t wLength

Number of bytes to transfer in data stage

4.5.34.2.8 wValue

uint16_t wValue

Request specific value

4.5.35 USB_STRING_DESCRIPTOR_struct Struct Reference

Structure with pointers to the standard USB descriptors.

4.5.35.1 Detailed Description

Structure with pointers to the standard USB descriptors.

#include <usb_protocol_headers.h>

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.36 USB_STRING_LANG_ID_DESCRIPTOR_struct Struct Reference

Structure for the USB string Language ID descriptor.

4.5.36.1 Detailed Description

Structure for the USB string Language ID descriptor.

#include <usb_protocol_headers.h>

4.5.36.1.1 Data Fields

- USB_DESCRIPTOR_HEADER_t header
- uint16_t id_array [LANG_ID_NUM]

4.5.36.2 Field Documentation

The documentation for this struct was generated from the following file:

source/source-files/usb_common/

usb_protocol_headers.h

4.5.36.2.1 header

USB_DESCRIPTOR_HEADER_t header

Descriptor type and size

4.5.36.2.2 id_array

uint16_t id_array[LANG_ID_NUM]

4.6 File Documentation

4.6.1 source/source-files/circular_buffer/circular_buffer.c File Reference

This file contains the implementation for a circular buffer.

```
#include "circular_buffer.h"
```

4.6.1.1 Functions

- BUFFER_RETURN_CODE_t CIRCBUF_Enqueue (CIRCULAR_BUFFER_t *buffer, uint8_t data)
  Adds input data to circular buffer if there is space available.
- BUFFER_RETURN_CODE_t CIRCBUF_Dequeue (CIRCULAR_BUFFER_t *buffer, uint8_t *data)
  Pulls data from the circular buffer if it's available.
- bool CIRCBUF_Empty (CIRCULAR_BUFFER_t *buffer)
  Checks if the circular buffer is empty.
- bool CIRCBUF_Full (CIRCULAR_BUFFER_t *buffer)
  Checks if the circular buffer is full.
- uint16_t CIRCBUF_FreeSpace (CIRCULAR_BUFFER_t *buffer)
  Returns the number of available bytes in the circular buffer.

4.6.1.2 Detailed Description

This file contains the implementation for a circular buffer.

CIRCULARBUFFER CDC Circular Buffer Source File

Version: USB Device Stack Driver Version 1.0.0

4.6.2 source/source-files/circular_buffer/circular_buffer.h File Reference

This file contains prototypes and datatypes for a circular buffer.

```
#include <stdint.h>
#include <stdbool.h>
#include <usb_common_elements.h>
```

4.6.2.1 Data structures

- struct CIRCULAR_BUFFER_struct

4.6.2.2 Functions

- BUFFER_RETURN_CODE_t CIRCBUF_Enqueue (CIRCULAR_BUFFER_t *buffer, uint8_t data)
  Adds input data to circular buffer if there is space available.
- BUFFER_RETURN_CODE_t CIRCBUF_Dequeue (CIRCULAR_BUFFER_t *buffer, uint8_t *data)
  Pulls data from the circular buffer if it's available.
- bool CIRCBUF_Empty (CIRCULAR_BUFFER_t *buffer)
  Checks if the circular buffer is empty.
- bool CIRCBUF_Full (CIRCULAR_BUFFER_t *buffer)
  Checks if the circular buffer is full.
- uint16_t CIRCBUF_FreeSpace (CIRCULAR_BUFFER_t *buffer)
  Returns the number of available bytes in the circular buffer.

4.6.2.3 Typedefs

- typedef enum BUFFER_RETURN_CODE_enum BUFFER_RETURN_CODE_t
- typedef struct CIRCULAR_BUFFER_struct CIRCULAR_BUFFER_t

4.6.2.4 Enumerations

- enum BUFFER_RETURN_CODE_enum { BUFFER_SUCCESS = 0, BUFFER_FULL = -1, BUFFER_EMPTY = -2 }

4.6.2.5 Detailed Description

This file contains prototypes and datatypes for a circular buffer.

CIRCULARBUFFER CDC Circular Buffer Header File

Version: USB Device Stack Driver Version 1.0.0

4.6.2.6 Typedef Documentation

4.6.2.6.1 BUFFER_RETURN_CODE_t

typedef enum BUFFER_RETURN_CODE_enum BUFFER_RETURN_CODE_t

4.6.2.6.2 CIRCULAR_BUFFER_t

typedef struct CIRCULAR_BUFFER_struct CIRCULAR_BUFFER_t

4.6.2.7 Enumeration Type Documentation

4.6.2.7.1 BUFFER_RETURN_CODE_enum

enum BUFFER_RETURN_CODE_enum

BUFFER_SUCCESS	Action successfully executed
BUFFER_FULL	Error triggered by full buffer
BUFFER_EMPTY	Error triggered by empty buffer

4.6.3 source/source-files/misra_project_deviations.txt File Reference

4.6.4 source/source-files/misra_specific_deviations.txt File Reference

4.6.5 source/source-files/misra_supported_rules.txt File Reference

4.6.6 source/source-files/usb_cdc.c File Reference

This file contains implementation for CDC.

```
#include <stddef.h>
#include <usb_cdc.h>
#include <usb_common_elements.h>
#include <usb_core.h>
#include <usb_core_requests.h>
#include <usb_core_transfer.h>
#include <usb_protocol_cdc.h>
#include <usb_protocol_headers.h>
#include <usb_config.h>
```

4.6.6.1 Functions

- void USB_CDCInitialize (void)
  Initializes the CDC class.
- RETURN_CODE_t USB_CDCRequestHandler (USB_SETUP_REQUEST_t *setupRequestPtr)
  Performs handling of control transfers.
- bool USB_CDCDataTerminalReady (void)

Checks if the Data Terminal Equipment bit has been set from the host.

- void USB_CDCSetBaud (uint16_t baud)
  Sets the data transfer baud rate for the CDC communication.
- uint32_t USB_CDCGetBaud (void)
  Gets the data transfer baud rate for the CDC communication.
- void USB_CDCSetStopBits (USB_CDC_LINE_CODING_STOP_BITS_t numStopBits)
  Sets the number of data transfer stop bits for the CDC communication.
- USB_CDC_LINE_CODING_STOP_BITS_t USB_CDCGetStopBits (void)
  Gets the number of data transfer stop bits for the CDC communication.
- void USB_CDCSetParity (USD_CDC_LINE_CODING_PARITY_t parity)
  Sets the data transfer parity for the CDC communication.
- USD_CDC_LINE_CODING_PARITY_t USB_CDCGetParity (void)
  Gets the data transfer parity for the CDC communication.
- void USB_CDCSetDataBits (USD_CDC_LINE_CODING_DATA_BITS_t numDataBits)
  Sets the number of data transfer data bits for the CDC communication.
- USD_CDC_LINE_CODING_DATA_BITS_t USB_CDCGetDataBits (void)
  Gets the number of data transfer data bits for the CDC communication.

4.6.6.2 Variables

- STATIC uint16_t usbCDCControlLineState
- STATIC USB_CDC_LINE_CODING_t usbCDCLineCoding

4.6.6.3 Detailed Description

This file contains implementation for CDC.

USBCDC CDC Source File

Version: USB Device Stack Driver Version 1.0.0

4.6.6.4 Variable Documentation

4.6.6.4.1 usbCDCControlLineState

STATIC uint16_t usbCDCControlLineState

4.6.6.4.2 usbCDCLineCoding

STATIC USB_CDC_LINE_CODING_t usbCDCLineCoding

4.6.7 source/source-files/usb_cdc.h File Reference

```
#include <usb_common_elements.h>
#include <usb_protocol_cdc.h>
#include <usb_protocol_headers.h>
```

4.6.7.1 Functions

- void USB_CDCInitialize (void)
  Initializes the CDC class.
- RETURN_CODE_t USB_CDCRequestHandler (USB_SETUP_REQUEST_t *setupRequestPtr)
  Performs handling of control transfers.
- bool USB_CDCDataTerminalReady (void)
  Checks if the Data Terminal Equipment bit has been set from the host.
- void USB_CDCSetBaud (uint16_t baud)

Sets the data transfer baud rate for the CDC communication.

- uint32_t USB_CDCGetBaud (void)
  Gets the data transfer baud rate for the CDC communication.
- void USB_CDCSetStopBits (USB_CDC_LINE_CODING_STOP_BITS_t numStopBits)
  Sets the number of data transfer stop bits for the CDC communication.
- USB_CDC_LINE_CODING_STOP_BITS_t USB_CDCGetStopBits (void)
  Gets the number of data transfer stop bits for the CDC communication.
- void USB_CDCSetParity (USD_CDC_LINE_CODING_PARITY_t parity)
  Sets the data transfer parity for the CDC communication.
- USD_CDC_LINE_CODING_PARITY_t USB_CDCGetParity (void)
  Gets the data transfer parity for the CDC communication.
- void USB_CDCSetDataBits (USD_CDC_LINE_CODING_DATA_BITS_t numDataBits)
  Sets the number of data transfer data bits for the CDC communication.
- USD_CDC_LINE_CODING_DATA_BITS_t USB_CDCGetDataBits (void)
  Gets the number of data transfer data bits for the CDC communication.

4.6.7.2 Detailed Description

USBCDC CDC Header File

4.6.8 source/source-files/usb_cdc_virtual_serial_port.c File Reference

This file contains implementation for a CDC application.

```
#include <usb_cdc_virtual_serial_port.h>
#include <usb_cdc.h>
#include <stddef.h>
#include <string.h>
#include <stdbool.h>
#include <usb_config.h>
#include <circular_buffer.h>
```

4.6.8.1 Functions

- void USB_CDCVirtualSerialPortInitialize (void)
  Initializes the CDC Virtual Serial Port functionality.
- RETURN_CODE_t USB_CDCVirtualSerialPortHandler (void)
  Performs Virtual Serial Port writes using the CDC class.
- CDC_RETURN_CODE_t USB_CDCRead (uint8_t *data)
  Pulls data from the CDC receive buffer.
- CDC_RETURN_CODE_t USB_CDCWrite (uint8_t data)
  Adds data to the CDC transmit buffer.
- bool USB_CDCTxBusy (void)
  Checks if the transmit buffer is full.
- void USB_CDCDataReceived (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)
  Callback function called after the USB OUT transaction started.
- void USB_CDCDataTransmitted (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)
  Callback function called after the USB IN transaction started.

4.6.8.2 Variables

- static bool zlpStateTX = true
- STATIC USB_PIPE_t CDCTxPipe
- STATIC USB_PIPE_t CDCRxPipe
- STATIC uint8_t usbCDCReceiveTempBuffer [USB_CDC_RX_PACKET_SIZE]
- STATIC uint8_t usbCDCReceiveArray [USB_CDC_RX_BUFFER_SIZE]
- STATIC CIRCULAR_BUFFER_t usbCDCReceiveBuffer
- STATIC uint8_t usbCDCTransmitArray [USB_CDC_TX_BUFFER_SIZE]
- STATIC CIRCULAR_BUFFER_t usbCDCTransmitBuffer

4.6.8.3 Detailed Description

This file contains implementation for a CDC application.

USBCDCVIRTUALSERIALPORT CDC Virtual Serial Port Source File

Version: USB Device Stack Driver Version 1.0.0

4.6.8.4 Variable Documentation

4.6.8.4.1 CDCRxPipe

STATIC USB_PIPE_t CDCRxPipe

Initial value:

```
= {
    .address = USB_CDC_BULK_EP_OUT,
    .direction = USB_EP_DIR_OUT,
}
```

4.6.8.4.2 CDCTxPipe

STATIC USB_PIPE_t CDCTxPipe

Initial value:

```
= {
    .address = USB_CDC_BULK_EP_IN,
    .direction = USB_EP_DIR_IN,
}
```

4.6.8.4.3 usbCDCReceiveArray

STATIC uint8_t usbCDCReceiveArray[USB_CDC_RX_BUFFER_SIZE]

4.6.8.4.4 usbCDCReceiveBuffer

STATIC CIRCULAR_BUFFER_t usbCDCReceiveBuffer

Initial value:

```
= {
    .content = usbCDCReceiveArray,
    .head = 0,
    .tail = 0,
    .maxLength = USB_CDC_RX_BUFFER_SIZE,
}
```

4.6.8.4.5 usbCDCReceiveTempBuffer

STATIC uint8_t usbCDCReceiveTempBuffer[USB_CDC_RX_PACKET_SIZE]

4.6.8.4.6 usbCDCTransmitArray

STATIC uint8_t usbCDCTransmitArray[USB_CDC_TX_BUFFER_SIZE]

4.6.8.4.7 usbCDCTransmitBuffer

STATIC CIRCULAR_BUFFER_t usbCDCTransmitBuffer

Initial value:

```
= {
    .content = usbCDCTransmitArray,
    .head = 0,
    .tail = 0,
    .maxLength = USB_CDC_TX_BUFFER_SIZE,
}
```

4.6.8.4.8 zlpStateTX

bool zlpStateTX = true[static]

4.6.9 source/source-files/usb_cdc_virtual_serial_port.h File Reference

This file contains prototypes and datatypes for a CDC application.

```
#include <stdint.h>
#include <stdbool.h>
#include <usb_core.h>
#include <usb_common_elements.h>
#include <usb_protocol_cdc.h>
```

4.6.9.1 Functions

- void USB_CDCVirtualSerialPortInitialize (void)
  Initializes the CDC Virtual Serial Port functionality.
- RETURN_CODE_t USB_CDCVirtualSerialPortHandler (void)
  Performs Virtual Serial Port writes using the CDC class.
- CDC_RETURN_CODE_t USB_CDCRead (uint8_t *data)
  Pulls data from the CDC receive buffer.
- CDC_RETURN_CODE_t USB_CDCWrite (uint8_t data)
  Adds data to the CDC transmit buffer.
- bool USB_CDCTxBusy (void)
  Checks if the transmit buffer is full.
- void USB_CDCDataTransmitted (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)
  Callback function called after the USB IN transaction started.
- void USB_CDCDataReceived (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)
  Callback function called after the USB OUT transaction started.

4.6.9.2 Typedefs

- typedef enum CDC_RETURN_CODE_enum CDC_RETURN_CODE_t

4.6.9.3 Enumerations

- enum CDC_RETURN_CODE_enum { CDC_SUCCESS = 0, CDC_BUFFER_FULL = -1, CDC_BUFFER_EMPTY = -2 }

4.6.9.4 Detailed Description

This file contains prototypes and datatypes for a CDC application.

USBCDCVIRTUALSERIAL CDC Virtual Serial Port Header File

Version: USB Device Stack Driver Version 1.0.0

4.6.9.5 Typedef Documentation

4.6.9.5.1 CDC_RETURN_CODE_t

typedef enum CDC_RETURN_CODE_enum CDC_RETURN_CODE_t

4.6.9.6 Enumeration Type Documentation

4.6.9.6.1 CDC_RETURN_CODE_enum

enum CDC_RETURN_CODE_enum

CDC_SUCCESS	Action successfully executed
CDC_BUFFER_FULL	Error triggered by full CDC buffer
CDC_BUFFER_EMPTY	Error triggered by empty CDC buffer

4.6.10 source/source-files/usb_common/usb_common_elements.h File Reference

4.6.10.1 Macros

- #define STATIC static
- #define MAX_ENDPOINT_SIZE_DEFAULT (64)
  The maximum endpoint packet size for the default endpoint types (control, bulk, interrupt).
- #define MAX_ENDPOINT_SIZE_ISO (1023)
  The maximum endpoint packet size for the isochronous endpoint type.

4.6.10.2 Typedefs

- typedef enum RETURN_CODE_enum RETURN_CODE_t

4.6.10.3 Enumerations

- enum RETURN_CODE_enum { UNSUPPORTED = 2, UNINITIALIZED = 1, SUCCESS = 0, ENDPOINT_SIZE_ERROR = -1, ENDPOINT_ADDRESS_ERROR = -2, ENDPOINT_DIRECTION_ERROR = -3, ENDPOINT_TYPE_ERROR = -4, ENDPOINT_AZLP_ERROR = -5, ENDPOINT_ALIGN_ERROR = -6, PIPE_BUSY_ERROR = -10, PIPE_TRANSFER_ERROR = -11, CONTROL_SIZE_ERROR = -20, CONTROL_TRANSACTION_STATUS_ERROR = -21, CONTROL_SETUP_CALLBACK_ERROR = -22, CONTROL_SETUP_DIRECTION_ERROR = -23, DESCRIPTOR_POINTER_ERROR = -30, DESCRIPTOR_CONFIGURATIONS_ERROR = -31, DESCRIPTOR_INTERFACE_ERROR = -32, DESCRIPTOR_ENDPOINT_ERROR = -33, DESCRIPTOR_REQUEST_ERROR = -34, DESCRIPTOR_SEARCH_ERROR = -35, INTERFACE_SET_ERROR = -40, INTERFACE_GET_ERROR = -41, USB_CONNECTION_ERROR = -50, USB_CLASS_ERROR = -60 }
  Describes the different function return reserved codes used by the USB stack.

4.6.10.4 Detailed Description

USBCOMMONELEMENTS Common Elements Header File

4.6.10.5 Macro Definition Documentation

4.6.10.5.1 STATIC

#define STATIC static

4.6.10.6 Typedef Documentation

4.6.10.6.1 RETURN_CODE_t

typedef enum RETURN_CODE_enum RETURN_CODE_t

4.6.11 source/source-files/usb_common/usb_core.c File Reference

```
#include <stdbool.h>
#include <stddef.h>
```

```
#include <stdint.h>
#include <string.h>
#include <usb_common_elements.h>
#include <usb_config.h>
#include <usb_core.h>
#include <usb_core_descriptors.h>
#include <usb_core_events.h>
#include <usb_core_requests.h>
#include <usb_core_transfer.h>
#include <usb_peripheral.h>
#include <usb_protocol_headers.h>
```

4.6.11.1 Functions

- RETURN_CODE_t USB_SetupProcess (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the Standard Device Request USB 2.0 Specification Ch 9.4.
- RETURN_CODE_t USB_Start (void)
  Starts the USB peripheral, configures the callbacks and attaches it to the bus.
- RETURN_CODE_t USB_Stop (void)
  Stops the USB peripheral and detaches it from the bus.
- RETURN_CODE_t USB_Reset (void)
  Resets the USB peripheral.

4.6.12 source/source-files/usb_common/usb_core.h File Reference

Core functionality for the USB stack.

```
#include <stdbool.h>
#include <stdint.h>
#include <usb_common_elements.h>
#include <usb_core_events.h>
#include <usb_core_transfer.h>
#include <usb_protocol_headers.h>
```

4.6.12.1 Functions

- RETURN_CODE_t USB_SetupProcess (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the Standard Device Request USB 2.0 Specification Ch 9.4.
- RETURN_CODE_t USB_Start (void)
  Starts the USB peripheral, configures the callbacks and attaches it to the bus.
- RETURN_CODE_t USB_Stop (void)
  Stops the USB peripheral and detaches it from the bus.
- RETURN_CODE_t USB_Reset (void)
  Resets the USB peripheral.

4.6.12.2 Detailed Description

Core functionality for the USB stack.

USBCORE CORE Source File

Version: USB Device Core Version 1.0.0
USBCORE CORE Header File

4.6.13 source/source-files/usb_common/usb_core_descriptors.c File Reference

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include <usb_common_elements.h>
#include <usb_config.h>
#include <usb_core.h>
```

```
#include <usb_core_descriptors.h>
#include <usb_core_events.h>
#include <usb_peripheral.h>
#include <usb_protocol_headers.h>
```

4.6.13.1 Data structures

- union USB_DESCRIPTOR_PTR_union

4.6.13.2 Functions

- RETURN_CODE_t USB_DescriptorPointersSet (USB_DESCRIPTOR_POINTERS_t *descriptorPointersPtr)

  Handles Descriptor pointer setup. Sets the address to the application descriptor pointers. Checks if the device pointer and a pointer to the start of the application configuration(s) are set before saving the address to the USB Core Stack.
- RETURN_CODE_t USB_DescriptorConfigurationEnable (uint8_t configurationValue)

  Enables endpoint configuration descriptor.
- uint8_t USB_DescriptorActiveConfigurationValueGet (void)

  Gets the active configuration value.
- bool USB_DescriptorActiveConfigurationSelfPoweredGet (void)

  Gets the Self-Powered setting from the active configuration.
- bool USB_DescriptorActiveConfigurationRemoteWakeupGet (void)

  Gets the Remote Wake-up setting from the active configuration.
- RETURN_CODE_t ConfigurationPointerGet (uint8_t configurationValue, USB_CONFIGURATION_DESCRIPTOR_t **configurationPtr)

  Collects the configuration pointer.
- RETURN_CODE_t NextDescriptorPointerGet (USB_DESCRIPTOR_TYPE_t descriptorType, USB_DESCRIPTOR_HEADER_t **descriptorHeaderPtr)

  Gets the next descriptor.
- RETURN_CODE_t USB_DescriptorInterfaceConfigure (uint8_t interfaceNumber, uint8_t alternateSetting, bool enable)

  Enables or Disables an Interface Descriptor.
- RETURN_CODE_t ActiveAlternateSettingSet (uint8_t interfaceNumber, uint8_t alternateSetting)

  Set the active alternate interface number for an interface.
- RETURN_CODE_t ActiveAlternateSettingGet (uint8_t interfaceNumber, uint8_t *alternateSetting)

  Get the active alternate interface number for an interface.
- RETURN_CODE_t DescriptorEndpointsConfigure (USB_INTERFACE_DESCRIPTOR_t *interfacePtr, bool enable)

  Configures the endpoints as given in the descriptor.
- RETURN_CODE_t USB_DescriptorPointerGet (USB_DESCRIPTOR_TYPE_t descriptor, uint8_t attribute, uint8_t **descriptorPtr, uint16_t *descriptorLength)

  Gets the pointer to the descriptor.
- RETURN_CODE_t USB_DescriptorStringPointerGet (uint8_t stringIndex, uint16_t langID, uint8_t **descriptorAddressPtr, uint16_t *descriptorLength)

  Gets the pointer to the string descriptor.

4.6.13.3 Macros

- #define USB_DEFAULT_INTERFACE 0u

  Default interface number.

- #define USB_DEFAULT_ALTERNATE_SETTING 0u
  Default alternate setting.
- #define USB_DESCRIPTOR_SEARCH_LIMIT 30u
  The number of descriptors NextDescriptorPointerGet will search through before returning an error.

4.6.13.4 Typedefs

- typedef union USB_DESCRIPTOR_PTR_union USB_DESCRIPTOR_PTR_t
  Union of a uint8_t pointer and pointers to the different descriptor types.

4.6.13.5 Variables

- STATIC USB_CONFIGURATION_DESCRIPTOR_t * activeConfigurationPtr = NULL
- STATIC uint8_t activeInterfaces [USB_INTERFACE_NUM]
- STATIC USB_DESCRIPTOR_POINTERS_t * applicationPointers = NULL

4.6.13.6 Typedef Documentation

4.6.13.6.1 USB_DESCRIPTOR_PTR_t

USB_DESCRIPTOR_PTR_t

Union of a uint8_t pointer and pointers to the different descriptor types.

MISRA C:2012 Deviation **Advisory:** misra-c2012-19.2
Justification: Needed for the stack to parse through the configuration descriptors without pointer casting between the different descriptor types and uint8_t.

4.6.13.7 Variable Documentation

4.6.13.7.1 activeConfigurationPtr

STATIC USB_CONFIGURATION_DESCRIPTOR_t* activeConfigurationPtr = NULL

4.6.13.7.2 activeInterfaces

STATIC uint8_t activeInterfaces[USB_INTERFACE_NUM]

4.6.13.7.3 applicationPointers

STATIC USB_DESCRIPTOR_POINTERS_t* applicationPointers = NULL

4.6.14 source/source-files/usb_common/usb_core_descriptors.h File Reference

descriptors for the USB Core Stack.

```
#include <stdbool.h>
#include <usb_common_elements.h>
#include <usb_protocol_headers.h>
```

4.6.14.1 Functions

- RETURN_CODE_t USB_DescriptorPointersSet (USB_DESCRIPTOR_POINTERS_t *descriptorPtr)
  Handles Descriptor pointer setup. Sets the address to the application descriptor pointers. Checks if the device pointer and a pointer to the start of the application configuration(s) are set before saving the address to the USB Core Stack.
- RETURN_CODE_t USB_DescriptorConfigurationEnable (uint8_t configurationValue)
  Enables endpoint configuration descriptor.
- bool USB_DescriptorActiveConfigurationSelfPoweredGet (void)
  Gets the Self-Powered setting from the active configuration.
- bool USB_DescriptorActiveConfigurationRemoteWakeupGet (void)
  Gets the Remote Wake-up setting from the active configuration.

- uint8_t USB_DescriptorActiveConfigurationValueGet (void)
  Gets the active configuration value.
- RETURN_CODE_t USB_DescriptorInterfaceConfigure (uint8_t interfaceNumber, uint8_t alternateSetting, bool enable)
  Enables or Disables an Interface Descriptor.
- RETURN_CODE_t USB_DescriptorPointerGet (USB_DESCRIPTOR_TYPE_t descriptor, uint8_t attribute, uint8_t **descriptorPtr, uint16_t *descriptorLength)
  Gets the pointer to the descriptor.
- RETURN_CODE_t USB_DescriptorStringPointerGet (uint8_t stringIndex, uint16_t langID, uint8_t **descriptorAddressPtr, uint16_t *descriptorLength)
  Gets the pointer to the string descriptor.
- RETURN_CODE_t ActiveAlternateSettingGet (uint8_t interfaceNumber, uint8_t *alternateSetting)
  Get the active alternate interface number for an interface.
- RETURN_CODE_t ActiveAlternateSettingSet (uint8_t interfaceNumber, uint8_t alternateSetting)
  Set the active alternate interface number for an interface.
- RETURN_CODE_t ConfigurationPointerGet (uint8_t configurationValue, USB_CONFIGURATION_DESCRIPTOR_t **configurationPtr)
  Collects the configuration pointer.
- RETURN_CODE_t DescriptorEndpointsConfigure (USB_INTERFACE_DESCRIPTOR_t *interfacePtr, bool enable)
  Configures the endpoints as given in the descriptor.
- RETURN_CODE_t NextDescriptorPointerGet (USB_DESCRIPTOR_TYPE_t descriptorType, USB_DESCRIPTOR_HEADER_t **descriptorHeaderPtr)
  Gets the next descriptor.

4.6.14.2 Detailed Description

descriptors for the USB Core Stack.

USBCOREDESCRIPTOR Core Descriptors Source File

Version: USB Device Core Version 1.0.0
USBCOREDESCRIPTOR Core Descriptors Header File

4.6.15 source/source-files/usb_common/usb_core_events.c File Reference

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include <usb_common_elements.h>
#include <usb_config.h>
#include <usb_core.h>
#include <usb_core_events.h>
#include <usb_peripheral.h>
#include <usb_protocol_headers.h>
```

4.6.15.1 Functions

- RETURN_CODE_t USB_EventHandler (void)
  Handles the different types of events.
- void USB_SetConfigurationCallbackRegister (USB_SETUP_EVENT_CALLBACK_t callback)
  Registers a callback for Set Configuration requests.
- void USB_SetInterfaceCallbackRegister (USB_SETUP_EVENT_CALLBACK_t callback)
  Registers a callback for Set Interface requests.

- void USB_InterfaceDisabledCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for disabling interfaces.
- void USB_VendorRequestCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)
  Registers a callback for vendor requests.
- void USB_ClassRequestCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)
  Registers a callback for class requests.
- void USB_OtherRequestCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)
  Registers a callback for other requests.
- void USB_SOFCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for Start-Of-Frame (SOF) events.
- void USB_ResetCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for Reset events.
- void USB_SuspendCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for Suspend events.
- void USB_ResumeCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for Resume events.

4.6.15.2 Variables

- USB_EVENT_HANDLERS_t event

4.6.15.3 Variable Documentation

4.6.15.3.1 event

USB_EVENT_HANDLERS_t event

4.6.16 source/source-files/usb_common/usb_core_events.h File Reference

Event handling for the USB Core Stack.

```
#include <stdbool.h>
#include <stdint.h>
#include <usb_common_elements.h>
#include <usb_core_descriptors.h>
#include <usb_core_requests.h>
#include <usb_core_transfer.h>
#include <usb_peripheral.h>
#include <usb_protocol_headers.h>
```

4.6.16.1 Data structures

- struct USB_EVENT_HANDLERS_struct
  Represents the event callbacks, the configuration and the enumeration setups.

4.6.16.2 Functions

- RETURN_CODE_t USB_EventHandler (void)
  Handles the different types of events.
- void USB_SetConfigurationCallbackRegister (USB_SETUP_EVENT_CALLBACK_t callback)
  Registers a callback for Set Configuration requests.
- void USB_SetInterfaceCallbackRegister (USB_SETUP_EVENT_CALLBACK_t callback)
  Registers a callback for Set Interface requests.
- void USB_InterfaceDisabledCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for disabling interfaces.

- void USB_VendorRequestCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)
  Registers a callback for vendor requests.
- void USB_ClassRequestCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)
  Registers a callback for class requests.
- void USB_OtherRequestCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)
  Registers a callback for other requests.
- void USB_SOFCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for Start-Of-Frame (SOF) events.
- void USB_ResetCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for Reset events.
- void USB_SuspendCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for Suspend events.
- void USB_ResumeCallbackRegister (USB_EVENT_CALLBACK_t callback)
  Registers a callback for Resume events.

4.6.16.3 Typedefs

- typedef struct USB_EVENT_HANDLERS_struct USB_EVENT_HANDLERS_t

4.6.16.4 Variables

- USB_EVENT_HANDLERS_t event

4.6.16.5 Detailed Description

Event handling for the USB Core Stack.

USBCOREEVENTS USB Core Events Source File

Version: USB Device Core Version 1.0.0
USBCOREEVENTS USB Core Events Header File

4.6.16.6 Typedef Documentation

4.6.16.6.1 USB_EVENT_HANDLERS_t

typedef struct USB_EVENT_HANDLERS_struct USB_EVENT_HANDLERS_t

4.6.16.7 Variable Documentation

4.6.16.7.1 event

USB_EVENT_HANDLERS_t event

4.6.17 source/source-files/usb_common/usb_core_requests.c File Reference

USB Device Core Requests handling.

```
#include <stdbool.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include <usb_core_requests.h>
#include <usb_common_elements.h>
#include <usb_protocol_headers.h>
#include <usb_config.h>
#include <usb_peripheral.h>
#include <usb_core.h>
```

4.6.17.1 Functions

- RETURN_CODE_t USB_SetupProcessDeviceRequest (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the device requests.

- RETURN_CODE_t USB_SetupProcessEndpointRequest (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the endpoint requests.
- RETURN_CODE_t USB_SetupProcessInterfaceRequest (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the interface requests.

4.6.17.2 Detailed Description

USB Device Core Requests handling.

USBCOREREQUESTS USB Core Requests Source File

Version: USB Device Core Version 1.0.0

4.6.18 source/source-files/usb_common/usb_core_requests.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include <usb_core_requests_device.h>
#include <usb_core_requests_interface.h>
#include <usb_core_requests_endpoint.h>
#include <usb_protocol_headers.h>
#include <usb_common_elements.h>
```

4.6.18.1 Functions

- RETURN_CODE_t USB_SetupProcessDeviceRequest (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the device requests.
- RETURN_CODE_t USB_SetupProcessEndpointRequest (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the endpoint requests.
- RETURN_CODE_t USB_SetupProcessInterfaceRequest (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the interface requests.

4.6.18.2 Detailed Description

USBCOREREQUESTS USB Core Requests Header File

4.6.19 source/source-files/usb_common/usb_core_requests_device.c File Reference

USB Device Core Requests handling.

```
#include <stdbool.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include <usb_core_requests_device.h>
#include <usb_common_elements.h>
#include <usb_protocol_headers.h>
#include <usb_config.h>
#include <usb_peripheral.h>
#include <usb_core.h>
#include <usb_core_descriptors.h>
```

4.6.19.1 Functions

- RETURN_CODE_t SetupDeviceRequestGetStatus (void)
  Returns the status of the device features.
- RETURN_CODE_t SetupDeviceRequestClearFeature (USB_SETUP_REQUEST_t *setupRequestPtr)
  Clears the device feature.
- RETURN_CODE_t SetupDeviceRequestSetFeature (USB_SETUP_REQUEST_t *setupRequestPtr)
  Sets the device feature.
- RETURN_CODE_t SetupDeviceRequestSetAddress (uint8_t address)
  Sets the device address.

- void SetupDeviceAddressCallback (void)
  Callback function for the address.
- RETURN_CODE_t SetupDeviceRequestGetDescriptor (USB_SETUP_REQUEST_t *setupRequestPtr)
  Gets the device descriptor.
- RETURN_CODE_t SetupDeviceRequestGetConfiguration (void)
  Gets the device configuration.
- RETURN_CODE_t SetupDeviceRequestSetConfiguration (uint8_t configurationValue)
  Sets the device configuration.

4.6.19.2 Variables

- STATIC uint8_t deviceAddress = 0

4.6.19.3 Detailed Description

USB Device Core Requests handling.

USBCOREREQUESTSDEVICE USB Core Requests Device Core File

Version: USB Device Core Version 1.0.0

4.6.19.4 Variable Documentation

4.6.19.4.1 deviceAddress

STATIC uint8_t deviceAddress = 0

4.6.20 source/source-files/usb_common/usb_core_requests_device.h File Reference

USB Device Core Requests handling.

```
#include <stdbool.h>
#include <stdint.h>
#include "usb_protocol_headers.h"
#include "usb_common_elements.h"
```

4.6.20.1 Functions

- RETURN_CODE_t SetupDeviceRequestGetStatus (void)
  Returns the status of the device features.
- RETURN_CODE_t SetupDeviceRequestClearFeature (USB_SETUP_REQUEST_t *setupRequestPtr)
  Clears the device feature.
- RETURN_CODE_t SetupDeviceRequestSetFeature (USB_SETUP_REQUEST_t *setupRequestPtr)
  Sets the device feature.
- RETURN_CODE_t SetupDeviceRequestSetAddress (uint8_t address)
  Sets the device address.
- void SetupDeviceAddressCallback (void)
  Callback function for the address.
- RETURN_CODE_t SetupDeviceRequestGetDescriptor (USB_SETUP_REQUEST_t *setupRequestPtr)
  Gets the device descriptor.
- RETURN_CODE_t SetupDeviceRequestGetConfiguration (void)
  Gets the device configuration.
- RETURN_CODE_t SetupDeviceRequestSetConfiguration (uint8_t configurationValue)
  Sets the device configuration.

4.6.20.2 Detailed Description

USB Device Core Requests handling.

USBCOREREQUESTSDEVICE USB Core Requests Device Header File

Version: USB Device Core Version 1.0.0

4.6.21 source/source-files/usb_common/usb_core_requests_endpoint.c File Reference

USB Endpoint Core Requests handling.

```
#include <stdbool.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include <usb_core_requests.h>
#include <usb_common_elements.h>
#include <usb_protocol_headers.h>
#include <usb_config.h>
#include <usb_peripheral.h>
#include <usb_core.h>
#include <usb_core_transfer.h>
```

4.6.21.1 Functions

- USB_PIPE_t EndpointFromRequestGet (uint16_t wIndex)
  Gets the endpoint status.
- RETURN_CODE_t SetupEndpointRequestGetStatus (USB_SETUP_REQUEST_t *setupRequestPtr)
  Gets the endpoint status.
- RETURN_CODE_t SetupEndpointRequestClearFeature (USB_SETUP_REQUEST_t *setupRequestPtr)
  Clears the endpoint feature.
- RETURN_CODE_t SetupEndpointRequestSetFeature (USB_SETUP_REQUEST_t *setupRequestPtr)
  Sets the endpoint feature.
- RETURN_CODE_t SetupEndpointRequestSynchFrame (void)
  Gets the current frame number.

4.6.21.2 Macros

- #define GET_STATUS_ENDPOINT_STALLED (1u << 0u)
  Mask for the endpoint stall status in the first byte of the data stage of the setup request.
- #define ENDPOINT_ADDRESS_MASK (0x7fu)
  Mask for the endpoint address in the wIndex field of the setup request.
- #define ENDPOINT_DIRECTION_BITPOSITION (7u)
  Bit position for the endpoint direction in the wIndex field of the setup request.

4.6.21.3 Detailed Description

USB Endpoint Core Requests handling.

USBCOREREQUESTSENDPOINT USB Core Requests Endpoint Source File

Version: USB Device Core Version USB 1.0.0

4.6.22 source/source-files/usb_common/usb_core_requests_endpoint.h File Reference

USB Endpoint Core Requests handling.

```
#include <stdbool.h>
#include <stdint.h>
#include "usb_protocol_headers.h"
#include "usb_common_elements.h"
```

4.6.22.1 Functions

- USB_PIPE_t EndpointFromRequestGet (uint16_t wIndex)

Gets the endpoint status.

- RETURN_CODE_t SetupEndpointRequestGetStatus (USB_SETUP_REQUEST_t *setupRequestPtr)
  Gets the endpoint status.
- RETURN_CODE_t SetupEndpointRequestClearFeature (USB_SETUP_REQUEST_t *setupRequestPtr)
  Clears the endpoint feature.
- RETURN_CODE_t SetupEndpointRequestSetFeature (USB_SETUP_REQUEST_t *setupRequestPtr)
  Sets the endpoint feature.
- RETURN_CODE_t SetupEndpointRequestSynchFrame (void)
  Gets the current frame number.

4.6.22.2 Detailed Description

USB Endpoint Core Requests handling.

USBCOREREQUESTSENDPOINT USB Core Requests Endpoint Header File

Version: USB Device Core Version USB 1.0.0

4.6.23 source/source-files/usb_common/usb_core_requests_interface.c File Reference

USB Interface Core Requests handling.

```
#include <stdbool.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include <usb_core_requests_interface.h>
#include <usb_common_elements.h>
#include <usb_protocol_headers.h>
#include <usb_config.h>
#include <usb_peripheral.h>
#include <usb_core.h>
#include <usb_core_events.h>
```

4.6.23.1 Functions

- RETURN_CODE_t USB_SetupInterfaceRequestGetStatus (void)
  Returns status for the specified interface.
- RETURN_CODE_t USB_SetupInterfaceRequestGetInterface (USB_SETUP_REQUEST_t *setupRequestPtr)
  Returns the alternate setting for the specified interface.
- RETURN_CODE_t USB_SetupInterfaceRequestSetInterface (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the interface request to select the alternate setting.
- RETURN_CODE_t USB_SetupInterfaceRequestGetDescriptor (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the interface request for class-specific descriptors.

4.6.23.2 Macros

- #define GET_INTERFACE_REQUEST_NUMBER_MASK (0xffu)
  Mask for the interface number in the wIndex field of the setup request.
- #define GET_INTERFACE_REQUEST_WVALUE 0u
  Value for the wValue field of the setup request.
- #define GET_INTERFACE_RESPONSE_SIZE 1u
  Size of the response to the Get Interface request.

4.6.23.3 Detailed Description

USB Interface Core Requests handling.

USBCOREREQUESTSINTERFACE USB Core Requests Interface Source File

Version: USB Device Core Version 1.0.0

4.6.24 source/source-files/usb_common/usb_core_requests_interface.h File Reference

USB Interface Core Requests handling.

```
#include <stdbool.h>
#include <stdint.h>
#include "usb_protocol_headers.h"
#include "usb_common_elements.h"
```

4.6.24.1 Functions

- RETURN_CODE_t USB_SetupInterfaceRequestGetStatus (void)
  Returns status for the specified interface.
- RETURN_CODE_t USB_SetupInterfaceRequestGetInterface (USB_SETUP_REQUEST_t *setupRequestPtr)
  Returns the alternate setting for the specified interface.
- RETURN_CODE_t USB_SetupInterfaceRequestSetInterface (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the interface request to select the alternate setting.
- RETURN_CODE_t USB_SetupInterfaceRequestGetDescriptor (USB_SETUP_REQUEST_t *setupRequestPtr)
  Setup function for the interface request for class-specific descriptors.

4.6.24.2 Detailed Description

USB Interface Core Requests handling.

USBCOREREQUESTSINTERFACE USB Core Requests Interface Header File

Version: USB Device Core Version 1.0.0

4.6.25 source/source-files/usb_common/usb_core_transfer.c File Reference

USB core layer implementation file.

```
#include <stdbool.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include <usb_core.h>
#include <usb_protocol_headers.h>
#include <usb_common_elements.h>
#include <usb_config.h>
#include <usb_peripheral.h>
```

4.6.25.1 Functions

- RETURN_CODE_t USB_TransferWriteStart (USB_PIPE_t pipe, uint8_t *dataPtr, uint16_t dataSize, bool useZLP, USB_TRANSFER_END_CALLBACK_t callback)
  Sets up the pipe for the write transfers.
- RETURN_CODE_t USB_TransferReadStart (USB_PIPE_t pipe, uint8_t *dataPtr, uint16_t dataSize, bool useZLP, USB_TRANSFER_END_CALLBACK_t callback)
  Sets up the pipe for the read transfers.
- RETURN_CODE_t USB_TransferControlDataSet (uint8_t *dataPtr, uint16_t dataSize, USB_SETUP_ENDOFREQUEST_CALLBACK_t callback)

Sets up vendor or class control request data transfers.

- RETURN_CODE_t USB_TransferAbort (USB_PIPE_t pipe)
  Aborts an ongoing transfer.
- RETURN_CODE_t USB_TransferHandler (void)
  Handles the different types of packages received or transferred.

4.6.25.2 Detailed Description

USB core layer implementation file.

USBCORETRANSFER USB Core Transfer Source File

Version: USB Core Version 1.0.0

4.6.26 source/source-files/usb_common/usb_core_transfer.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include <usb_protocol_headers.h>
#include <usb_common_elements.h>
#include <usb_core_transfer.h>
#include <usb_core_requests.h>
#include <usb_core_descriptors.h>
#include <usb_peripheral.h>
```

4.6.26.1 Functions

- RETURN_CODE_t USB_TransferWriteStart (USB_PIPE_t pipe, uint8_t *dataPtr, uint16_t dataSize, bool useZLP, USB_TRANSFER_END_CALLBACK_t callback)
  Sets up the pipe for the write transfers.
- RETURN_CODE_t USB_TransferReadStart (USB_PIPE_t pipe, uint8_t *dataPtr, uint16_t dataSize, bool useZLP, USB_TRANSFER_END_CALLBACK_t callback)
  Sets up the pipe for the read transfers.
- RETURN_CODE_t USB_TransferControlDataSet (uint8_t *dataPtr, uint16_t dataSize, USB_SETUP_ENDOFREQUEST_CALLBACK_t callback)
  Sets up vendor or class control request data transfers.
- RETURN_CODE_t USB_TransferAbort (USB_PIPE_t pipe)
  Aborts an ongoing transfer.
- RETURN_CODE_t USB_TransferHandler (void)
  Handles the different types of packages received or transferred.

4.6.26.2 Detailed Description

USBCORETRANSFER USB Core Transfer Header File

4.6.27 source/source-files/usb_common/usb_protocol_headers.h File Reference

```
#include "usb_common_elements.h"
#include <stdbool.h>
#include <stdint.h>
#include <usb_config.h>
```

4.6.27.1 Data structures

- struct USB_PIPE_struct
  Structure for a pipe which has an address and direction.
- struct USB_SETUP_REQUEST_struct
- struct USB_DESCRIPTOR_HEADER_struct

- struct USB_DEVICE_DESCRIPTOR_struct
- struct USB_DEV_QUAL_DESC_struct
- struct USB_DEV_BOS_DESC_struct
- struct USB_DEV_CAPA_EXT_DESC_struct
- struct USB_DEV_LPM_DESC_struct
- struct USB_ASSOCIATION_DESC_struct
- struct USB_CONFIGURATION_DESCRIPTOR_struct
- struct USB_IAD_DESC_struct
  Standard USB association descriptor structure.
- struct USB_INTERFACE_DESCRIPTOR_struct
  Ch. 9.6.5 Standard USB interface descriptor structure.
- struct USB_ENDPOINT_DESCRIPTOR_struct
- struct USB_STRING_LANG_ID_DESCRIPTOR_struct
  Structure for the USB string Language ID descriptor.
- struct USB_DESCRIPTOR_POINTERS_struct

4.6.27.2 Macros

- #define USB_EP_DIR_IN 0x01u
  Endpoint direction IN.
- #define USB_EP_DIR_OUT 0x00u
  Endpoint direction OUT.
- #define OVERFLOW_EVENT 1u
  Overflow event for the endpoint.
- #define UNDERFLOW_EVENT 2u
  Underflow event for the endpoint.
- #define USB_V2_0 0x0200
  USB Specification version 2.00.
- #define USB_V2_1 0x0201
  USB Specification version 2.01.
- #define NO_SUBCLASS 0x00
  No subclass code.
- #define NO_PROTOCOL 0x00
  No protocol code.
- #define CLASS_IAD 0xEF
  IAD class code.
- #define SUB_CLASS_IAD 0x02
  IAD subclass code.
- #define PROTOCOL_IAD 0x01
  IAD protocol code.
- #define USB_ENDPOINT_FEATURE_HALT 0x00u
  USB endpoint feature halt.
- #define DESCRIPTOR_STRING_LENGTH(wstring) (sizeof(wstring) / sizeof(wchar_t) - 1)
  Calculates descriptor length of a UTF-16 string descriptor without the null character.

- #define USB_CONFIG_ATTR_MUST_SET (1u << 7u)
  USB Attribute bitfield for the configuration descriptor.
- #define USB_CONFIG_ATTR_BUS_POWERED (0u << 6u)
  USB Attribute Bus Powered bitfield for the configuration descriptor.
- #define USB_CONFIG_ATTR_SELF_POWERED (1u << 6u)
  USB Attribute Self Powered bitfield for the configuration descriptor.
- #define USB_CONFIG_ATTR_REMOTE_WAKEUP (1u << 5u)
  USB Attribute Remote Wakeup bitfield for the configuration descriptor.
- #define USB_CONFIG_MAX_POWER(ma) (((ma) + 1u) / 2u)
  USB Max Power bitfield for the configuration descriptor.
- #define USB_REQUEST_DEVICE_SELF_POWERED (1u << 0u)
- #define USB_REQUEST_DEVICE_REMOTE_WAKEUP (1u << 1u)
- #define USB_REQUEST_DEVICE_DISABLE_CONFIGURATION (0u)

4.6.27.3 Typedefs

- typedef enum USB_ENDPOINT_enum USB_ENDPOINT_t
- typedef struct USB_PIPE_struct USB_PIPE_t
- typedef enum USB_TRANSFER_STATUS_enum USB_TRANSFER_STATUS_t
- typedef enum USB_CONTROL_STATUS_enum USB_CONTROL_STATUS_t
- typedef enum USB_DEVICE_CLASS_enum USB_DEVICE_CLASS_t
- typedef enum USB_REQUEST_DIR_enum USB_REQUEST_DIR_t
- typedef enum USB_REQUEST_TYPE_enum USB_REQUEST_TYPE_t
- typedef enum USB_REQUEST_RECIPIENT_enum USB_REQUEST_RECIPIENT_t
- typedef enum USB_REQUEST_ID_enum USB_REQUEST_ID_t
- typedef enum USB_DESCRIPTOR_TYPE_enum USB_DESCRIPTOR_TYPE_t
- typedef struct USB_SETUP_REQUEST_struct USB_SETUP_REQUEST_t
- typedef struct USB_DESCRIPTOR_HEADER_struct USB_DESCRIPTOR_HEADER_t
- typedef struct USB_DEVICE_DESCRIPTOR_struct USB_DEVICE_DESCRIPTOR_t
- typedef struct USB_DEV_QUAL_DESC_struct USB_DEV_QUAL_DESC_t
- typedef struct USB_DEV_BOS_DESC_struct USB_DEV_BOS_DESC_t
- typedef struct USB_DEV_CAPA_EXT_DESC_struct USB_DEV_CAPA_EXT_DESC_t
- typedef struct USB_DEV_LPM_DESC_struct USB_DEV_LPM_DESC_t
- typedef struct USB_ASSOCIATION_DESC_struct USB_ASSOCIATION_DESC_t
- typedef struct USB_CONFIGURATION_DESCRIPTOR_struct USB_CONFIGURATION_DESCRIPTOR_t
- typedef struct USB_IAD_DESC_struct USB_IAD_DESC_t
- typedef struct USB_INTERFACE_DESCRIPTOR_struct USB_INTERFACE_DESCRIPTOR_t
- typedef struct USB_ENDPOINT_DESCRIPTOR_struct USB_ENDPOINT_DESCRIPTOR_t
- typedef struct USB_STRING_LANG_ID_DESCRIPTOR_struct USB_STRING_LANG_ID_DESCRIPTOR_t
- typedef struct USB_DESCRIPTOR_POINTERS_struct USB_DESCRIPTOR_POINTERS_t
- typedef void(* USB_TRANSFER_END_CALLBACK_t) (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)
  Function callback type USB_TRANSFER_END_CALLBACK_t. Callback type used for transfer complete notifications.

- typedef RETURN_CODE_t(* USB_SETUP_PROCESS_CALLBACK_t) (USB_SETUP_REQUEST_t *setupRequestPtr)

  Function callback type USB_SETUP_PROCESS_CALLBACK_t. Callback type used for setup request processing, with a return code to let the stack know to proceed.
- typedef RETURN_CODE_t(* USB_SETUP_STRING_CALLBACK_t) (uint8_t stringIndex, uint16_t langID, uint8_t **descriptorAddressPtr, uint16_t *descriptorLength)

  Function callback type USB_SETUP_STRING_CALLBACK_t. Callback type used for setup request processing a string descriptor, with a return code to let the stack know to proceed.
- typedef void(* USB_SETUP_EVENT_CALLBACK_t) (USB_SETUP_REQUEST_t *setupRequestPtr)

  Function callback type USB_SETUP_EVENT_CALLBACK_t. Callback type used for setup request notifications.
- typedef RETURN_CODE_t(* USB_SETUP_OVERUNDERRUN_CALLBACK_t) (void)

  Function callback type USB_SETUP_OVERUNDERRUN_CALLBACK_t. Callback type used for USB Overrun and Underrun event processing on the control endpoints, with a return code to let the stack know to proceed.
- typedef void(* USB_SETUP_ENDOFREQUEST_CALLBACK_t) (void)

  Function callback type USB_SETUP_ENDOFREQUEST_CALLBACK_t. Callback type used for setup request complete notifications.
- typedef void(* USB_EVENT_CALLBACK_t) (void)

  Function callback type USB_EVENT_CALLBACK_t. Callback type used for USB event notifications.

4.6.27.4 Enumerations

- enum USB_ENDPOINT_enum { CONTROL = 0, ISOCHRONOUS = 1, BULK = 2, INTERRUPT = 3, DISABLED = 0xff }

  Defines labels for the different endpoint types as per the USB 2.0 base specification.
- enum USB_TRANSFER_STATUS_enum { USB_PIPE_TRANSFER_OK = 0, USB_PIPE_TRANSFER_BUSY = 1, USB_PIPE_TRANSFER_ABORTED = 2, USB_PIPE_TRANSFER_ERROR = 3 }

  Defines the possible states of a configured transfer.
- enum USB_CONTROL_STATUS_enum { USB_CONTROL_SETUP = 0, USB_CONTROL_DATA_OUT = 1, USB_CONTROL_DATA_IN = 2, USB_CONTROL_ZLP = 3, USB_CONTROL_STALL_REQ = 4 }

  Defines the possible states of a configured control transfer.
- enum USB_DEVICE_CLASS_enum { USB_NO_DEVICE_CLASS = 0x00, USB_AUDIO_DEVICE_CLASS = 0x01, USB_CDC_DEVICE_CLASS = 0x02, USB_HID_DEVICE_CLASS = 0x03, USB_PHYSICAL_DEVICE_CLASS = 0x05, USB_IMAGE_DEVICE_CLASS = 0x06, USB_PRINTER_DEVICE_CLASS = 0x07, USB_MASS_STORAGE_DEVICE_CLASS = 0x08, USB_HUB_DEVICE_CLASS = 0x09, USB_CDC_DATA_DEVICE_CLASS = 0x0A, USB_SMART_CARD_DEVICE_CLASS = 0x0B, USB_CONTENT_SECURITY_DEVICE_CLASS = 0x0D, USB_VIDEO_DEVICE_CLASS = 0x0E, USB_PERSONAL_HEALTHCARE_DEVICE_CLASS = 0x0F, USB_AUDIO_VIDEO_DEVICE_CLASS = 0x10, USB_BILLBOARD_DEVICE_CLASS = 0x11, USB_TYPE_C_BRIDGE_DEVICE_CLASS = 0x12, USB_BULK_DISPLAY_PROTOCOL_DEVICE_CLASS = 0x13, USB_MCTP_DEVICE_CLASS = 0x14, USB_I3C_DEVICE_CLASS = 0x3C, USB_DIAGNOSTIC_DEVICE_CLASS = 0xDC, USB_WIRELESS_DEVICE_CLASS = 0xE0, USB_MISC_DEVICE_CLASS = 0xEF, USB_APPLICATION_DEVICE_CLASS = 0xFE, USB_VENDOR_DEVICE_CLASS = 0xFF }
- enum USB_REQUEST_DIR_enum { USB_REQUEST_DIR_OUT = 0, USB_REQUEST_DIR_IN = 1 }

  Standard USB enumeration used by setup requests.
- enum USB_REQUEST_TYPE_enum { USB_REQUEST_TYPE_STANDARD = 0, USB_REQUEST_TYPE_CLASS = 1, USB_REQUEST_TYPE_VENDOR = 2 }

  USB request types (bmRequestType).

- enum USB_REQUEST_RECIPIENT_enum { USB_REQUEST_RECIPIENT_DEVICE = 0, USB_REQUEST_RECIPIENT_INTERFACE = 1, USB_REQUEST_RECIPIENT_ENDPOINT = 2, USB_REQUEST_RECIPIENT_OTHER = 3 }

  USB recipient codes (bmRequestType).
- enum USB_REQUEST_ID_enum { USB_REQUEST_GET_STATUS = 0, USB_REQUEST_CLEAR_FEATURE = 1, USB_REQUEST_SET_FEATURE = 3, USB_REQUEST_SET_ADDRESS = 5, USB_REQUEST_GET_DESCRIPTOR = 6, USB_REQUEST_SET_DESCRIPTOR = 7, USB_REQUEST_GET_CONFIGURATION = 8, USB_REQUEST_SET_CONFIGURATION = 9, USB_REQUEST_GET_INTERFACE = 10, USB_REQUEST_SET_INTERFACE = 11, USB_REQUEST_SYNCH_FRAME = 12 }

  Standard USB requests (bRequest).
- enum USB_DESCRIPTOR_TYPE_enum { USB_DESCRIPTOR_TYPE_DEVICE = 1, USB_DESCRIPTOR_TYPE_CONFIGURATION = 2, USB_DESCRIPTOR_TYPE_STRING = 3, USB_DESCRIPTOR_TYPE_INTERFACE = 4, USB_DESCRIPTOR_TYPE_ENDPOINT = 5, USB_DESCRIPTOR_TYPE_DEVICE_QUALIFIER = 6, USB_DESCRIPTOR_TYPE_OTHER_SPEED_CONFIGURATION = 7, USB_DESCRIPTOR_TYPE_INTERFACE_POWER = 8, USB_DESCRIPTOR_TYPE_IAD = 11, USB_DESCRIPTOR_TYPE_BOS = 15, USB_DESCRIPTOR_TYPE_DEVICE_CAPABILITY = 16, USB_DESCRIPTOR_TYPE_CLASS = 0x20, USB_DESCRIPTOR_TYPE_VENDOR = 0x40 }

  Standard USB descriptor types.

4.6.27.5 Detailed Description

USBPROTOCOLHEADERS USB Protocol Headers Header File

4.6.27.6 Macro Definition Documentation

4.6.27.6.1 USB_REQUEST_DEVICE_DISABLE_CONFIGURATION

#define USB_REQUEST_DEVICE_DISABLE_CONFIGURATION (0u)

4.6.27.6.2 USB_REQUEST_DEVICE_REMOTE_WAKEUP

#define USB_REQUEST_DEVICE_REMOTE_WAKEUP (1u << 1u)

Remote Wake-up supported

4.6.27.6.3 USB_REQUEST_DEVICE_SELF_POWERED

#define USB_REQUEST_DEVICE_SELF_POWERED (1u << 0u)

Self-Powered

4.6.27.7 Typedef Documentation

4.6.27.7.1 USB_ASSOCIATION_DESC_t

typedef struct USB_ASSOCIATION_DESC_struct USB_ASSOCIATION_DESC_t

4.6.27.7.2 USB_CONFIGURATION_DESCRIPTOR_t

typedef struct USB_CONFIGURATION_DESCRIPTOR_struct USB_CONFIGURATION_DESCRIPTOR_t

4.6.27.7.3 USB_CONTROL_STATUS_t

typedef enum USB_CONTROL_STATUS_enum USB_CONTROL_STATUS_t

4.6.27.7.4 USB_DESCRIPTOR_HEADER_t

typedef struct USB_DESCRIPTOR_HEADER_struct USB_DESCRIPTOR_HEADER_t

4.6.27.7.5 USB_DESCRIPTOR_POINTERS_t

typedef struct USB_DESCRIPTOR_POINTERS_struct USB_DESCRIPTOR_POINTERS_t

4.6.27.7.6 USB_DESCRIPTOR_TYPE_t

typedef enum USB_DESCRIPTOR_TYPE_enum USB_DESCRIPTOR_TYPE_t

4.6.27.7.7 USB_DEV_BOS_DESC_t
typedef struct USB_DEV_BOS_DESC_struct USB_DEV_BOS_DESC_t

4.6.27.7.8 USB_DEV_CAPA_EXT_DESC_t
typedef struct USB_DEV_CAPA_EXT_DESC_struct USB_DEV_CAPA_EXT_DESC_t

4.6.27.7.9 USB_DEV_LPM_DESC_t
typedef struct USB_DEV_LPM_DESC_struct USB_DEV_LPM_DESC_t

4.6.27.7.10 USB_DEV_QUAL_DESC_t
typedef struct USB_DEV_QUAL_DESC_struct USB_DEV_QUAL_DESC_t

4.6.27.7.11 USB_DEVICE_CLASS_t
typedef enum USB_DEVICE_CLASS_enum USB_DEVICE_CLASS_t

4.6.27.7.12 USB_DEVICE_DESCRIPTOR_t
typedef struct USB_DEVICE_DESCRIPTOR_struct USB_DEVICE_DESCRIPTOR_t

4.6.27.7.13 USB_ENDPOINT_DESCRIPTOR_t
typedef struct USB_ENDPOINT_DESCRIPTOR_struct USB_ENDPOINT_DESCRIPTOR_t

4.6.27.7.14 USB_ENDPOINT_t
typedef enum USB_ENDPOINT_enum USB_ENDPOINT_t

4.6.27.7.15 USB_IAD_DESC_t
typedef struct USB_IAD_DESC_struct USB_IAD_DESC_t

4.6.27.7.16 USB_INTERFACE_DESCRIPTOR_t
typedef struct USB_INTERFACE_DESCRIPTOR_struct USB_INTERFACE_DESCRIPTOR_t

4.6.27.7.17 USB_PIPE_t
typedef struct USB_PIPE_struct USB_PIPE_t

4.6.27.7.18 USB_REQUEST_DIR_t
typedef enum USB_REQUEST_DIR_enum USB_REQUEST_DIR_t

4.6.27.7.19 USB_REQUEST_ID_t
typedef enum USB_REQUEST_ID_enum USB_REQUEST_ID_t

4.6.27.7.20 USB_REQUEST_RECIPIENT_t
typedef enum USB_REQUEST_RECIPIENT_enum USB_REQUEST_RECIPIENT_t

4.6.27.7.21 USB_REQUEST_TYPE_t
typedef enum USB_REQUEST_TYPE_enum USB_REQUEST_TYPE_t

4.6.27.7.22 USB_SETUP_REQUEST_t
typedef struct USB_SETUP_REQUEST_struct USB_SETUP_REQUEST_t

4.6.27.7.23 USB_STRING_LANG_ID_DESCRIPTOR_t
typedef struct USB_STRING_LANG_ID_DESCRIPTOR_struct USB_STRING_LANG_ID_DESCRIPTOR_t

4.6.27.7.24 USB_TRANSFER_STATUS_t
typedef enum USB_TRANSFER_STATUS_enum USB_TRANSFER_STATUS_t

4.6.27.8 Enumeration Type Documentation

4.6.27.8.1 USB_DEVICE_CLASS_enum
enum USB_DEVICE_CLASS_enum

USB_NO_DEVICE_CLASS	Use class code info from Interface Descriptors

USB_AUDIO_DEVICE_CLASS	Audio device
USB_CDC_DEVICE_CLASS	Communications and CDC Control
USB_HID_DEVICE_CLASS	HID (Human Interface Device)
USB_PHYSICAL_DEVICE_CLASS	Physical device
USB_IMAGE_DEVICE_CLASS	Still imaging device
USB_PRINTER_DEVICE_CLASS	Printer device
USB_MASS_STORAGE_DEVICE_CLASS	Mass storage device
USB_HUB_DEVICE_CLASS	Hub
USB_CDC_DATA_DEVICE_CLASS	CDC data device
USB_SMART_CARD_DEVICE_CLASS	Smart Card device
USB_CONTENT_SECURITY_DEVICE_CLASS	Content security device
USB_VIDEO_DEVICE_CLASS	Video device
USB_PERSONAL_HEALTHCARE_DEVICE_CLASS	Personal healthcare device
USB_AUDIO_VIDEO_DEVICE_CLASS	Audio/Video devices
USB_BILLBOARD_DEVICE_CLASS	Billboard device
USB_TYPE_C_BRIDGE_DEVICE_CLASS	USB Type-C bridge device
USB_BULK_DISPLAY_PROTOCOL_DEVICE_CLASS	USB Bulk display protocol device
USB_MCTP_DEVICE_CLASS	MCTP over USB protocol endpoint device
USB_I3C_DEVICE_CLASS	I3C device
USB_DIAGNOSTIC_DEVICE_CLASS	Diagnostic device
USB_WIRELESS_DEVICE_CLASS	Wireless controller
USB_MISC_DEVICE_CLASS	Miscellaneous
USB_APPLICATION_DEVICE_CLASS	Application specific
USB_VENDOR_DEVICE_CLASS	Vendor specific

4.6.28 source/source-files/usb_hid.c File Reference

Contains the implementation for the HID generic drivers.

```
#include <usb_hid.h>
#include <stddef.h>
#include <usb_common_elements.h>
#include <usb_protocol_hid.h>
#include <usb_core.h>
#include <usb_config.h>
```

4.6.28.1 Functions

- void USB_HIDReportUpdatedCallbackRegister (USB_HID_REPORT_CALLBACK_t callback)
  Registers a callback to the HID report.
- void USB_HIDReportUpdatedCallback (void)
  Checks if a callback is registered and calls the End Of Request function.
- void USB_HIDInitialize (uint8_t *ratePtr, uint8_t *protocolPtr, USB_HID_REPORT_DESCRIPTOR_t *reportPtr)
  Registers the rate, protocol and report descriptor for HID.
- RETURN_CODE_t USB_HIDRequestHandler (USB_SETUP_REQUEST_t *setupRequestPtr)
  Initializes the HID class and performs control transfers.

4.6.28.2 Variables

- STATIC uint8_t * reportDescriptor = NULL
- STATIC uint8_t * rate = NULL
- STATIC uint8_t * protocol = NULL
- STATIC uint16_t reportData = 0
- STATIC uint8_t * descriptorPtr = NULL
- STATIC uint16_t descriptorLength = 0
- STATIC USB_DESCRIPTOR_TYPE_HID_t descriptorType
- STATIC USB_HID_REPORT_CALLBACK_t reportCallback = NULL

4.6.28.3 Detailed Description

Contains the implementation for the HID generic drivers.

USBHID HID Source File

Version: USB Device Stack HID Driver Version 1.0.0

4.6.28.4 Variable Documentation

4.6.28.4.1 descriptorLength

STATIC uint16_t descriptorLength = 0

4.6.28.4.2 descriptorPtr

STATIC uint8_t* descriptorPtr = NULL

4.6.28.4.3 descriptorType

STATIC USB_DESCRIPTOR_TYPE_HID_t descriptorType

4.6.28.4.4 protocol

STATIC uint8_t* protocol = NULL

4.6.28.4.5 rate

STATIC uint8_t* rate = NULL

4.6.28.4.6 reportCallback

STATIC USB_HID_REPORT_CALLBACK_t reportCallback = NULL

4.6.28.4.7 reportData

STATIC uint16_t reportData = 0

4.6.28.4.8 reportDescriptor

STATIC uint8_t* reportDescriptor = NULL

4.6.29 source/source-files/usb_hid.h File Reference

```
#include <usb_common_elements.h>
#include <usb_protocol_headers.h>
#include <usb_protocol_hid.h>
```

4.6.29.1 Functions

- void USB_HIDReportUpdatedCallbackRegister (USB_HID_REPORT_CALLBACK_t callback)
  Registers a callback to the HID report.
- void USB_HIDReportUpdatedCallback (void)
  Checks if a callback is registered and calls the End Of Request function.
- void USB_HIDInitialize (uint8_t *ratePtr, uint8_t *protocolPtr, USB_HID_REPORT_DESCRIPTOR_t *reportPtr)

Registers the rate, protocol and report descriptor for HID.

- RETURN_CODE_t USB_HIDRequestHandler (USB_SETUP_REQUEST_t *setupRequestPtr)
  Initializes the HID class and performs control transfers.

4.6.29.2 Detailed Description

USBHID HID Header File

4.6.30 source/source-files/usb_hid_keyboard.c File Reference

Contains the implementation for the USB Keyboard drivers.

```
#include <usb_hid_keyboard.h>
#include <usb_hid_keycodes.h>
#include <stddef.h>
#include <string.h>
#include <usb_common_elements.h>
#include <usb_hid.h>
#include <usb_hid_transfer.h>
#include <usb_core.h>
#include <usb_config.h>
#include <usb_protocol_hid.h>
```

4.6.30.1 Functions

- void USB_HIDKeyboardInitialize (USB_HID_REPORT_DESCRIPTOR_t *reportPtr, USB_HID_REPORT_CALLBACK_t callback)
  Sets up the keyboard application for use with the HID class.
- RETURN_CODE_t USB_HIDKeyModifierDown (uint8_t modifierID)
  Updates the keyboard report by adding the pressed modifier key in the modifier byte in the report array.
- RETURN_CODE_t USB_HIDKeyModifierUp (uint8_t modifierID)
  Updates the keyboard report by removing the released modifier key in the modifier byte in the report array.
- int8_t USB_HIDKeyCodeIndexGet (uint8_t keyID)
  Checks the report array to see that the key is already present.
- RETURN_CODE_t USB_HIDKeyPressDown (uint8_t keyID)
  Updates the keyboard report by adding the pressed key at the first available byte in the report array.
- RETURN_CODE_t USB_HIDKeyPressUp (uint8_t keyID)
  Updates the keyboard report by removing the released key and shifts the registered keys towards the beginning of the report array.

4.6.30.2 Variables

- STATIC uint8_t usbHIDKeyboardRate
- STATIC uint8_t usbHIDKeyboardProtocol
- STATIC USB_KEYBOARD_REPORT_DATA_t keyboardInputReport

4.6.30.3 Detailed Description

Contains the implementation for the USB Keyboard drivers.

USBHIDKEYBOARD HID Keyboard Source File

Version: USB Device Stack HID Driver Version 1.0.0

4.6.30.4 Variable Documentation

4.6.30.4.1 keyboardInputReport

STATIC USB_KEYBOARD_REPORT_DATA_t keyboardInputReport

4.6.30.4.2 usbHIDKeyboardProtocol

STATIC uint8_t usbHIDKeyboardProtocol

4.6.30.4.3 usbHIDKeyboardRate

STATIC uint8_t usbHIDKeyboardRate

4.6.31 source/source-files/usb_hid_keyboard.h File Reference

```
#include <usb_protocol_hid.h>
#include <usb_common_elements.h>
```

4.6.31.1 Keyboard input report sizes

Macros for the input report for a standard HID keyboard.

- #define USB_HID_KEYBOARD_REPORT_KEYNUM 6u
- #define USB_HID_KEYBOARD_REPORT_SIZE (USB_HID_KEYBOARD_REPORT_KEYNUM + 2u)
- void USB_HIDKeyboardInitialize (USB_HID_REPORT_DESCRIPTOR_t *reportPtr, USB_HID_REPORT_CALLBACK_t callback)

  Sets up the keyboard application for use with the HID class.
- RETURN_CODE_t USB_HIDKeyModifierDown (uint8_t modifierID)

  Updates the keyboard report by adding the pressed modifier key in the modifier byte in the report array.
- RETURN_CODE_t USB_HIDKeyModifierUp (uint8_t modifierID)

  Updates the keyboard report by removing the released modifier key in the modifier byte in the report array.
- int8_t USB_HIDKeyCodeIndexGet (uint8_t keyID)

  Checks the report array to see that the key is already present.
- RETURN_CODE_t USB_HIDKeyPressDown (uint8_t keyID)

  Updates the keyboard report by adding the pressed key at the first available byte in the report array.
- RETURN_CODE_t USB_HIDKeyPressUp (uint8_t keyID)

  Updates the keyboard report by removing the released key and shifts the registered keys towards the beginning of the report array.

4.6.31.2 Detailed Description

USBHIDKEYBOARD HID Keyboard Header File

4.6.32 source/source-files/usb_hid_keycodes.h File Reference

USB Human Interface Device (HID) keycode definitions.

4.6.32.1 Macros

4.6.32.1.1 HID keyboard key codes

Macros for the key codes used in HID applications

- #define HID_KEYID_NOT_FOUND (-1)
- #define HID_KEY_NONE 0x00
- #define HID_ERROR_ROLLOVER 0x01
- #define HID_POST_FAIL 0x02
- #define HID_ERROR_UNDEFINED 0x03
- #define HID_A 0x04

- #define HID_B 0x05
- #define HID_C 0x06
- #define HID_D 0x07
- #define HID_E 0x08
- #define HID_F 0x09
- #define HID_G 0x0A
- #define HID_H 0x0B
- #define HID_I 0x0C
- #define HID_J 0x0D
- #define HID_K 0x0E
- #define HID_L 0x0F
- #define HID_M 0x10
- #define HID_N 0x11
- #define HID_O 0x12
- #define HID_P 0x13
- #define HID_Q 0x14
- #define HID_R 0x15
- #define HID_S 0x16
- #define HID_T 0x17
- #define HID_U 0x18
- #define HID_V 0x19
- #define HID_W 0x1A
- #define HID_X 0x1B
- #define HID_Y 0x1C
- #define HID_Z 0x1D
- #define HID_1 0x1E
- #define HID_2 0x1F
- #define HID_3 0x20
- #define HID_4 0x21
- #define HID_5 0x22
- #define HID_6 0x23
- #define HID_7 0x24
- #define HID_8 0x25
- #define HID_9 0x26
- #define HID_0 0x27
- #define HID_RETURN 0x28
- #define HID_ESCAPE 0x29
- #define HID_BACKSPACE 0x2A
- #define HID_TAB 0x2B
- #define HID_SPACEBAR 0x2C
- #define HID_UNDERSCORE 0x2D

- #define HID_EQUAL 0x2E
- #define HID_OPEN_BRACE 0x2F
- #define HID_CLOSE_BRACE 0x30
- #define HID_BACKSLASH 0x31
- #define HID_HASH_TILDE 0x32
- #define HID_SEMICOLON 0x33
- #define HID_APOSTROPHE 0x34
- #define HID_GRAVE_TILDE 0x35
- #define HID_COMMA 0x36
- #define HID_DOT 0x37
- #define HID_SLASH 0x38
- #define HID_CAPS_LOCK 0x39
- #define HID_F1 0x3A
- #define HID_F2 0x3B
- #define HID_F3 0x3C
- #define HID_F4 0x3D
- #define HID_F5 0x3E
- #define HID_F6 0x3F
- #define HID_F7 0x40
- #define HID_F8 0x41
- #define HID_F9 0x42
- #define HID_F10 0x43
- #define HID_F11 0x44
- #define HID_F12 0x45
- #define HID_F13 0x68
- #define HID_F14 0x69
- #define HID_F15 0x6A
- #define HID_F16 0x6B
- #define HID_F17 0x6C
- #define HID_F18 0x6D
- #define HID_F19 0x6E
- #define HID_F20 0x6F
- #define HID_F21 0x70
- #define HID_F22 0x71
- #define HID_F23 0x72
- #define HID_F24 0x73
- #define HID_PRINTSCREEN 0x46
- #define HID_SCROLL_LOCK 0x47
- #define HID_PAUSE 0x48
- #define HID_INSERT 0x49
- #define HID_HOME 0x4A

- #define HID_PAGEUP 0x4B
- #define HID_DELETE 0x4C
- #define HID_END 0x4D
- #define HID_PAGEDOWN 0x4E
- #define HID_RIGHT 0x4F
- #define HID_LEFT 0x50
- #define HID_DOWN 0x51
- #define HID_UP 0x52
- #define HID_KEYPAD_NUM_LOCK 0x53
- #define HID_KEYPAD_SLASH 0x54
- #define HID_KEYPAD_ASTERISK 0x55
- #define HID_KEYPAD_MINUS 0x56
- #define HID_KEYPAD_PLUS 0x57
- #define HID_KEYPAD_ENTER 0x58
- #define HID_KEYPAD_1 0x59
- #define HID_KEYPAD_2 0x5A
- #define HID_KEYPAD_3 0x5B
- #define HID_KEYPAD_4 0x5C
- #define HID_KEYPAD_5 0x5D
- #define HID_KEYPAD_6 0x5E
- #define HID_KEYPAD_7 0x5F
- #define HID_KEYPAD_8 0x60
- #define HID_KEYPAD_9 0x61
- #define HID_KEYPAD_0 0x62
- #define HID_KEYPAD_DOT 0x63
- #define HID_KEYPAD_EQUAL 0x67
- #define HID_KEYPAD_COMMA 0x85
- #define HID_KEYPAD_EQUALSIGN 0x86
- #define HID_KEYPAD_00 0xB0
- #define HID_KEYPAD_000 0xB1
- #define HID_KEYPAD_LEFT_PARENTHESIS 0xB6
- #define HID_KEYPAD_RIGHT_PARENTHESIS 0xB7
- #define HID_KEYPAD_OPEN_BRACE 0xB8
- #define HID_KEYPAD_CLOSE_BRACE 0xB9
- #define HID_KEYPAD_TAB 0xBA
- #define HID_KEYPAD_BACKSPACE 0xBB
- #define HID_KEYPAD_A 0xBC
- #define HID_KEYPAD_B 0xBD
- #define HID_KEYPAD_C 0xBE
- #define HID_KEYPAD_D 0xBF
- #define HID_KEYPAD_E 0xC0

- #define HID_KEYPAD_F 0xC1
- #define HID_KEYPAD_XOR 0xC2
- #define HID_KEYPAD_POWER_TO 0xC3
- #define HID_KEYPAD_PERCENT 0xC4
- #define HID_KEYPAD_LEFT_ANGLE_BRACE 0xC5
- #define HID_KEYPAD_RIGHT_ANGLE_BRACE 0xC6
- #define HID_KEYPAD_AND 0xC7
- #define HID_KEYPAD_AND_AND 0xC8
- #define HID_KEYPAD_OR 0xC9
- #define HID_KEYPAD_OR_OR 0xCA
- #define HID_KEYPAD_COLON 0xCB
- #define HID_KEYPAD_HASH 0xCC
- #define HID_KEYPAD_SPACE 0xCD
- #define HID_KEYPAD_AT 0xCE
- #define HID_KEYPAD_EXCLAMATION 0xCF
- #define HID_KEYPAD_MEM_STORE 0xD0
- #define HID_KEYPAD_MEM_RECALL 0xD1
- #define HID_KEYPAD_MEM_CLEAR 0xD2
- #define HID_KEYPAD_MEM_ADD 0xD3
- #define HID_KEYPAD_MEM_SUBTRACT 0xD4
- #define HID_KEYPAD_MEM_MULTIPLY 0xD5
- #define HID_KEYPAD_MEM_DIVIDE 0xD6
- #define HID_KEYPAD_PLUS_MINUS 0xD7
- #define HID_KEYPAD_CLEAR 0xD8
- #define HID_KEYPAD_CLEAR_ENTRY 0xD9
- #define HID_KEYPAD_BINARY 0xDA
- #define HID_KEYPAD_OCTAL 0xDB
- #define HID_KEYPAD_DECIMAL 0xDC
- #define HID_KEYPAD_HEXADECIMAL 0xDD
- #define HID_AT102 0x64
- #define HID_APPLICATION 0x65
- #define HID_POWER 0x66
- #define HID_EXECUTE 0x74
- #define HID_HELP 0x75
- #define HID_MENU 0x76
- #define HID_SELECT 0x77
- #define HID_STOP 0x78
- #define HID_AGAIN 0x79
- #define HID_UNDO 0x7A
- #define HID_CUT 0x7B
- #define HID_COPY 0x7C

- #define HID_PASTE 0x7D
- #define HID_FIND 0x7E
- #define HID_MUTE 0x7F
- #define HID_VOLUME_UP 0x80
- #define HID_VOLUME_DOWN 0x81
- #define HID_LOCK_CAPS_LOCK 0x82
- #define HID_LOCK_NUM_LOCK 0x83
- #define HID_LOCK_SCROLL_LOCK 0x84
- #define HID_INTERNATIONAL_1 0x87
- #define HID_INTERNATIONAL_2 0x88
- #define HID_INTERNATIONAL_3 0x89
- #define HID_INTERNATIONAL_4 0x8A
- #define HID_INTERNATIONAL_5 0x8B
- #define HID_INTERNATIONAL_6 0x8C
- #define HID_INTERNATIONAL_7 0x8D
- #define HID_INTERNATIONAL_8 0x8E
- #define HID_INTERNATIONAL_9 0x8F
- #define HID_LANG_1 0x90
- #define HID_LANG_2 0x91
- #define HID_LANG_3 0x92
- #define HID_LANG_4 0x93
- #define HID_LANG_5 0x94
- #define HID_LANG_6 0x95
- #define HID_LANG_7 0x96
- #define HID_LANG_8 0x97
- #define HID_LANG_9 0x98
- #define HID_KB_ALT_ERASE 0x99
- #define HID_KB_SYSREQ 0x9A
- #define HID_KB_CANCEL 0x9B
- #define HID_KB_CLEAR 0x9C
- #define HID_KB_PRIOR 0x9D
- #define HID_KB_RETURN 0x9E
- #define HID_KB_SEPARATOR 0x9F
- #define HID_KB_OUT 0xA0
- #define HID_KB_OPER 0xA1
- #define HID_KB_CLEAR_AGAIN 0xA2
- #define HID_KB_CRSEL 0xA3
- #define HID_KB_EXSEL 0xA4
- #define HID_1K_SEPARATOR 0xB2
- #define HID_DECIMAL_SEPARATOR 0xB3
- #define HID_CURRENCY_UNIT 0xB4

- #define HID_CURRENCY_SUB_UNIT 0xB5
- #define HID_LEFT_CTRL 0xE0
- #define HID_LEFT_SHIFT 0xE1
- #define HID_LEFT_ALT 0xE2
- #define HID_LEFT_GUI 0xE3
- #define HID_RIGHT_CTRL 0xE4
- #define HID_RIGHT_SHIFT 0xE5
- #define HID_RIGHT_ALT 0xE6
- #define HID_RIGHT_GUI 0xE7
- #define HID_MEDIA_PLAYPAUSE 0xE8
- #define HID_MEDIA_STOPCD 0xE9
- #define HID_MEDIA_PREVIOUSSONG 0xEA
- #define HID_MEDIA_NEXTSONG 0xEB
- #define HID_MEDIA_EJECTCD 0xEC
- #define HID_MEDIA_VOLUMEUP 0xED
- #define HID_MEDIA_VOLUMEDOWN 0xEE
- #define HID_MEDIA_MUTE 0xEF
- #define HID_MEDIA_WWW 0xF0
- #define HID_MEDIA_BACK 0xF1
- #define HID_MEDIA_FORWARD 0xF2
- #define HID_MEDIA_STOP 0xF3
- #define HID_MEDIA_FIND 0xF4
- #define HID_MEDIA_SCROLLUP 0xF5
- #define HID_MEDIA_SCROLLDOWN 0xF6
- #define HID_MEDIA_EDIT 0xF7
- #define HID_MEDIA_SLEEP 0xF8
- #define HID_MEDIA_COFFEE 0xF9
- #define HID_MEDIA_REFRESH 0xFA
- #define HID_MEDIA_CALC 0xFB

4.6.32.2 HID modifier key codes

Macros for the modifier keys

- #define HID_MODIFIER_NONE 0x00
- #define HID_MODIFIER_LEFT_CTRL 0x01
- #define HID_MODIFIER_LEFT_SHIFT 0x02
- #define HID_MODIFIER_LEFT_ALT 0x04
- #define HID_MODIFIER_LEFT_UI 0x08
- #define HID_MODIFIER_RIGHT_CTRL 0x10
- #define HID_MODIFIER_RIGHT_SHIFT 0x20
- #define HID_MODIFIER_RIGHT_ALT 0x40
- #define HID_MODIFIER_RIGHT_UI 0x80

4.6.32.3 HID LED codes

Macros for the LEDs

- #define HID_LED_NUM_LOCK (1 << 0)
- #define HID_LED_CAPS_LOCK (1 << 1)
- #define HID_LED_SCROLL_LOCK (1 << 2)
- #define HID_LED_COMPOSE (1 << 3)
- #define HID_LED_KANA (1 << 4)

4.6.32.4 Detailed Description

USB Human Interface Device (HID) keycode definitions.

USBHIDKEYCODES HID Keycodes Header File

Version: USB Device Stack HID Driver Version 1.0.0

4.6.32.5 Macro Definition Documentation

4.6.32.5.1 HID_0

#define HID_0 0x27

4.6.32.5.2 HID_1

#define HID_1 0x1E

4.6.32.5.3 HID_1K_SEPARATOR

#define HID_1K_SEPARATOR 0xB2

4.6.32.5.4 HID_2

#define HID_2 0x1F

4.6.32.5.5 HID_3

#define HID_3 0x20

4.6.32.5.6 HID_4

#define HID_4 0x21

4.6.32.5.7 HID_5

#define HID_5 0x22

4.6.32.5.8 HID_6

#define HID_6 0x23

4.6.32.5.9 HID_7

#define HID_7 0x24

4.6.32.5.10 HID_8

#define HID_8 0x25

4.6.32.5.11 HID_9

#define HID_9 0x26

4.6.32.5.12 HID_A

#define HID_A 0x04

4.6.32.5.13 HID_AGAIN

#define HID_AGAIN 0x79

4.6.32.5.14 HID_APOSTROPHE

#define HID_APOSTROPHE 0x34

4.6.32.5.15 HID_APPLICATION
#define HID_APPLICATION 0x65

4.6.32.5.16 HID_AT102
#define HID_AT102 0x64

4.6.32.5.17 HID_B
#define HID_B 0x05

4.6.32.5.18 HID_BACKSLASH
#define HID_BACKSLASH 0x31

4.6.32.5.19 HID_BACKSPACE
#define HID_BACKSPACE 0x2A

4.6.32.5.20 HID_C
#define HID_C 0x06

4.6.32.5.21 HID_CAPS_LOCK
#define HID_CAPS_LOCK 0x39

4.6.32.5.22 HID_CLOSE_BRACE
#define HID_CLOSE_BRACE 0x30

4.6.32.5.23 HID_COMMA
#define HID_COMMA 0x36

4.6.32.5.24 HID_COPY
#define HID_COPY 0x7C

4.6.32.5.25 HID_CURRENCY_SUB_UNIT
#define HID_CURRENCY_SUB_UNIT 0xB5

4.6.32.5.26 HID_CURRENCY_UNIT
#define HID_CURRENCY_UNIT 0xB4

4.6.32.5.27 HID_CUT
#define HID_CUT 0x7B

4.6.32.5.28 HID_D
#define HID_D 0x07

4.6.32.5.29 HID_DECIMAL_SEPARATOR
#define HID_DECIMAL_SEPARATOR 0xB3

4.6.32.5.30 HID_DELETE
#define HID_DELETE 0x4C

4.6.32.5.31 HID_DOT
#define HID_DOT 0x37

4.6.32.5.32 HID_DOWN
#define HID_DOWN 0x51

4.6.32.5.33 HID_E
#define HID_E 0x08

4.6.32.5.34 HID_END
#define HID_END 0x4D

4.6.32.5.35 HID_EQUAL
#define HID_EQUAL 0x2E

4.6.32.5.36 HID_ERROR_ROLLOVER

#define HID_ERROR_ROLLOVER 0x01

4.6.32.5.37 HID_ERROR_UNDEFINED

#define HID_ERROR_UNDEFINED 0x03

4.6.32.5.38 HID_ESCAPE

#define HID_ESCAPE 0x29

4.6.32.5.39 HID_EXECUTE

#define HID_EXECUTE 0x74

4.6.32.5.40 HID_F

#define HID_F 0x09

4.6.32.5.41 HID_F1

#define HID_F1 0x3A

4.6.32.5.42 HID_F10

#define HID_F10 0x43

4.6.32.5.43 HID_F11

#define HID_F11 0x44

4.6.32.5.44 HID_F12

#define HID_F12 0x45

4.6.32.5.45 HID_F13

#define HID_F13 0x68

4.6.32.5.46 HID_F14

#define HID_F14 0x69

4.6.32.5.47 HID_F15

#define HID_F15 0x6A

4.6.32.5.48 HID_F16

#define HID_F16 0x6B

4.6.32.5.49 HID_F17

#define HID_F17 0x6C

4.6.32.5.50 HID_F18

#define HID_F18 0x6D

4.6.32.5.51 HID_F19

#define HID_F19 0x6E

4.6.32.5.52 HID_F2

#define HID_F2 0x3B

4.6.32.5.53 HID_F20

#define HID_F20 0x6F

4.6.32.5.54 HID_F21

#define HID_F21 0x70

4.6.32.5.55 HID_F22

#define HID_F22 0x71

4.6.32.5.56 HID_F23

#define HID_F23 0x72

4.6.32.5.57 HID_F24
#define HID_F24 0x73

4.6.32.5.58 HID_F3
#define HID_F3 0x3C

4.6.32.5.59 HID_F4
#define HID_F4 0x3D

4.6.32.5.60 HID_F5
#define HID_F5 0x3E

4.6.32.5.61 HID_F6
#define HID_F6 0x3F

4.6.32.5.62 HID_F7
#define HID_F7 0x40

4.6.32.5.63 HID_F8
#define HID_F8 0x41

4.6.32.5.64 HID_F9
#define HID_F9 0x42

4.6.32.5.65 HID_FIND
#define HID_FIND 0x7E

4.6.32.5.66 HID_G
#define HID_G 0x0A

4.6.32.5.67 HID_GRAVE_TILDE
#define HID_GRAVE_TILDE 0x35

4.6.32.5.68 HID_H
#define HID_H 0x0B

4.6.32.5.69 HID_HASH_TILDE
#define HID_HASH_TILDE 0x32

4.6.32.5.70 HID_HELP
#define HID_HELP 0x75

4.6.32.5.71 HID_HOME
#define HID_HOME 0x4A

4.6.32.5.72 HID_I
#define HID_I 0x0C

4.6.32.5.73 HID_INSERT
#define HID_INSERT 0x49

4.6.32.5.74 HID_INTERNATIONAL_1
#define HID_INTERNATIONAL_1 0x87

4.6.32.5.75 HID_INTERNATIONAL_2
#define HID_INTERNATIONAL_2 0x88

4.6.32.5.76 HID_INTERNATIONAL_3
#define HID_INTERNATIONAL_3 0x89

4.6.32.5.77 HID_INTERNATIONAL_4
#define HID_INTERNATIONAL_4 0x8A

4.6.32.5.78 HID_INTERNATIONAL_5
#define HID_INTERNATIONAL_5 0x8B

4.6.32.5.79 HID_INTERNATIONAL_6
#define HID_INTERNATIONAL_6 0x8C

4.6.32.5.80 HID_INTERNATIONAL_7
#define HID_INTERNATIONAL_7 0x8D

4.6.32.5.81 HID_INTERNATIONAL_8
#define HID_INTERNATIONAL_8 0x8E

4.6.32.5.82 HID_INTERNATIONAL_9
#define HID_INTERNATIONAL_9 0x8F

4.6.32.5.83 HID_J
#define HID_J 0x0D

4.6.32.5.84 HID_K
#define HID_K 0x0E

4.6.32.5.85 HID_KB_ALT_ERASE
#define HID_KB_ALT_ERASE 0x99

4.6.32.5.86 HID_KB_CANCEL
#define HID_KB_CANCEL 0x9B

4.6.32.5.87 HID_KB_CLEAR
#define HID_KB_CLEAR 0x9C

4.6.32.5.88 HID_KB_CLEAR_AGAIN
#define HID_KB_CLEAR_AGAIN 0xA2

4.6.32.5.89 HID_KB_CRSEL
#define HID_KB_CRSEL 0xA3

4.6.32.5.90 HID_KB_EXSEL
#define HID_KB_EXSEL 0xA4

4.6.32.5.91 HID_KB_OPER
#define HID_KB_OPER 0xA1

4.6.32.5.92 HID_KB_OUT
#define HID_KB_OUT 0xA0

4.6.32.5.93 HID_KB_PRIOR
#define HID_KB_PRIOR 0x9D

4.6.32.5.94 HID_KB_RETURN
#define HID_KB_RETURN 0x9E

4.6.32.5.95 HID_KB_SEPARATOR
#define HID_KB_SEPARATOR 0x9F

4.6.32.5.96 HID_KB_SYSREQ
#define HID_KB_SYSREQ 0x9A

4.6.32.5.97 HID_KEY_NONE
#define HID_KEY_NONE 0x00

4.6.32.5.98 HID_KEYID_NOT_FOUND
#define HID_KEYID_NOT_FOUND (-1)

4.6.32.5.99 HID_KEYPAD_0
#define HID_KEYPAD_0 0x62

4.6.32.5.100 HID_KEYPAD_00
#define HID_KEYPAD_00 0xB0

4.6.32.5.101 HID_KEYPAD_000
#define HID_KEYPAD_000 0xB1

4.6.32.5.102 HID_KEYPAD_1
#define HID_KEYPAD_1 0x59

4.6.32.5.103 HID_KEYPAD_2
#define HID_KEYPAD_2 0x5A

4.6.32.5.104 HID_KEYPAD_3
#define HID_KEYPAD_3 0x5B

4.6.32.5.105 HID_KEYPAD_4
#define HID_KEYPAD_4 0x5C

4.6.32.5.106 HID_KEYPAD_5
#define HID_KEYPAD_5 0x5D

4.6.32.5.107 HID_KEYPAD_6
#define HID_KEYPAD_6 0x5E

4.6.32.5.108 HID_KEYPAD_7
#define HID_KEYPAD_7 0x5F

4.6.32.5.109 HID_KEYPAD_8
#define HID_KEYPAD_8 0x60

4.6.32.5.110 HID_KEYPAD_9
#define HID_KEYPAD_9 0x61

4.6.32.5.111 HID_KEYPAD_A
#define HID_KEYPAD_A 0xBC

4.6.32.5.112 HID_KEYPAD_AND
#define HID_KEYPAD_AND 0xC7

4.6.32.5.113 HID_KEYPAD_AND_AND
#define HID_KEYPAD_AND_AND 0xC8

4.6.32.5.114 HID_KEYPAD_ASTERISK
#define HID_KEYPAD_ASTERISK 0x55

4.6.32.5.115 HID_KEYPAD_AT
#define HID_KEYPAD_AT 0xCE

4.6.32.5.116 HID_KEYPAD_B
#define HID_KEYPAD_B 0xBD

4.6.32.5.117 HID_KEYPAD_BACKSPACE
#define HID_KEYPAD_BACKSPACE 0xBB

4.6.32.5.118 HID_KEYPAD_BINARY
#define HID_KEYPAD_BINARY 0xDA

4.6.32.5.119 HID_KEYPAD_C
#define HID_KEYPAD_C 0xBE

4.6.32.5.120 HID_KEYPAD_CLEAR
#define HID_KEYPAD_CLEAR 0xD8

4.6.32.5.121 HID_KEYPAD_CLEAR_ENTRY
#define HID_KEYPAD_CLEAR_ENTRY 0xD9

4.6.32.5.122 HID_KEYPAD_CLOSE_BRACE
#define HID_KEYPAD_CLOSE_BRACE 0xB9

4.6.32.5.123 HID_KEYPAD_COLON
#define HID_KEYPAD_COLON 0xCB

4.6.32.5.124 HID_KEYPAD_COMMA
#define HID_KEYPAD_COMMA 0x85

4.6.32.5.125 HID_KEYPAD_D
#define HID_KEYPAD_D 0xBF

4.6.32.5.126 HID_KEYPAD_DECIMAL
#define HID_KEYPAD_DECIMAL 0xDC

4.6.32.5.127 HID_KEYPAD_DOT
#define HID_KEYPAD_DOT 0x63

4.6.32.5.128 HID_KEYPAD_E
#define HID_KEYPAD_E 0xC0

4.6.32.5.129 HID_KEYPAD_ENTER
#define HID_KEYPAD_ENTER 0x58

4.6.32.5.130 HID_KEYPAD_EQUAL
#define HID_KEYPAD_EQUAL 0x67

4.6.32.5.131 HID_KEYPAD_EQUALSIGN
#define HID_KEYPAD_EQUALSIGN 0x86

4.6.32.5.132 HID_KEYPAD_EXCLAMATION
#define HID_KEYPAD_EXCLAMATION 0xCF

4.6.32.5.133 HID_KEYPAD_F
#define HID_KEYPAD_F 0xC1

4.6.32.5.134 HID_KEYPAD_HASH
#define HID_KEYPAD_HASH 0xCC

4.6.32.5.135 HID_KEYPAD_HEXADECIMAL
#define HID_KEYPAD_HEXADECIMAL 0xDD

4.6.32.5.136 HID_KEYPAD_LEFT_ANGLE_BRACE
#define HID_KEYPAD_LEFT_ANGLE_BRACE 0xC5

4.6.32.5.137 HID_KEYPAD_LEFT_PARENTHESIS
#define HID_KEYPAD_LEFT_PARENTHESIS 0xB6

4.6.32.5.138 HID_KEYPAD_MEM_ADD
#define HID_KEYPAD_MEM_ADD 0xD3

4.6.32.5.139 HID_KEYPAD_MEM_CLEAR
#define HID_KEYPAD_MEM_CLEAR 0xD2

4.6.32.5.140 HID_KEYPAD_MEM_DIVIDE
#define HID_KEYPAD_MEM_DIVIDE 0xD6

4.6.32.5.141 HID_KEYPAD_MEM_MULTIPLY
#define HID_KEYPAD_MEM_MULTIPLY 0xD5

4.6.32.5.142 HID_KEYPAD_MEM_RECALL
#define HID_KEYPAD_MEM_RECALL 0xD1

4.6.32.5.143 HID_KEYPAD_MEM_STORE
#define HID_KEYPAD_MEM_STORE 0xD0

4.6.32.5.144 HID_KEYPAD_MEM_SUBTRACT
#define HID_KEYPAD_MEM_SUBTRACT 0xD4

4.6.32.5.145 HID_KEYPAD_MINUS
#define HID_KEYPAD_MINUS 0x56

4.6.32.5.146 HID_KEYPAD_NUM_LOCK
#define HID_KEYPAD_NUM_LOCK 0x53

4.6.32.5.147 HID_KEYPAD_OCTAL
#define HID_KEYPAD_OCTAL 0xDB

4.6.32.5.148 HID_KEYPAD_OPEN_BRACE
#define HID_KEYPAD_OPEN_BRACE 0xB8

4.6.32.5.149 HID_KEYPAD_OR
#define HID_KEYPAD_OR 0xC9

4.6.32.5.150 HID_KEYPAD_OR_OR
#define HID_KEYPAD_OR_OR 0xCA

4.6.32.5.151 HID_KEYPAD_PERCENT
#define HID_KEYPAD_PERCENT 0xC4

4.6.32.5.152 HID_KEYPAD_PLUS
#define HID_KEYPAD_PLUS 0x57

4.6.32.5.153 HID_KEYPAD_PLUS_MINUS
#define HID_KEYPAD_PLUS_MINUS 0xD7

4.6.32.5.154 HID_KEYPAD_POWER_TO
#define HID_KEYPAD_POWER_TO 0xC3

4.6.32.5.155 HID_KEYPAD_RIGHT_ANGLE_BRACE
#define HID_KEYPAD_RIGHT_ANGLE_BRACE 0xC6

4.6.32.5.156 HID_KEYPAD_RIGHT_PARENTHESIS
#define HID_KEYPAD_RIGHT_PARENTHESIS 0xB7

4.6.32.5.157 HID_KEYPAD_SLASH
#define HID_KEYPAD_SLASH 0x54

4.6.32.5.158 HID_KEYPAD_SPACE
#define HID_KEYPAD_SPACE 0xCD

4.6.32.5.159 HID_KEYPAD_TAB
#define HID_KEYPAD_TAB 0xBA

4.6.32.5.160 HID_KEYPAD_XOR
#define HID_KEYPAD_XOR 0xC2

4.6.32.5.161 HID_L
#define HID_L 0x0F

4.6.32.5.162 HID_LANG_1
#define HID_LANG_1 0x90

4.6.32.5.163 HID_LANG_2
#define HID_LANG_2 0x91

4.6.32.5.164 HID_LANG_3
#define HID_LANG_3 0x92

4.6.32.5.165 HID_LANG_4
#define HID_LANG_4 0x93

4.6.32.5.166 HID_LANG_5
#define HID_LANG_5 0x94

4.6.32.5.167 HID_LANG_6
#define HID_LANG_6 0x95

4.6.32.5.168 HID_LANG_7
#define HID_LANG_7 0x96

4.6.32.5.169 HID_LANG_8
#define HID_LANG_8 0x97

4.6.32.5.170 HID_LANG_9
#define HID_LANG_9 0x98

4.6.32.5.171 HID_LED_CAPS_LOCK
#define HID_LED_CAPS_LOCK (1 << 1)

4.6.32.5.172 HID_LED_COMPOSE
#define HID_LED_COMPOSE (1 << 3)

4.6.32.5.173 HID_LED_KANA
#define HID_LED_KANA (1 << 4)

4.6.32.5.174 HID_LED_NUM_LOCK
#define HID_LED_NUM_LOCK (1 << 0)

4.6.32.5.175 HID_LED_SCROLL_LOCK
#define HID_LED_SCROLL_LOCK (1 << 2)

4.6.32.5.176 HID_LEFT
#define HID_LEFT 0x50

4.6.32.5.177 HID_LEFT_ALT
#define HID_LEFT_ALT 0xE2

4.6.32.5.178 HID_LEFT_CTRL
#define HID_LEFT_CTRL 0xE0

4.6.32.5.179 HID_LEFT_GUI
#define HID_LEFT_GUI 0xE3

4.6.32.5.180 HID_LEFT_SHIFT
#define HID_LEFT_SHIFT 0xE1

4.6.32.5.181 HID_LOCK_CAPS_LOCK
#define HID_LOCK_CAPS_LOCK 0x82

4.6.32.5.182 HID_LOCK_NUM_LOCK
#define HID_LOCK_NUM_LOCK 0x83

4.6.32.5.183 HID_LOCK_SCROLL_LOCK
#define HID_LOCK_SCROLL_LOCK 0x84

4.6.32.5.184 HID_M
#define HID_M 0x10

4.6.32.5.185 HID_MEDIA_BACK
#define HID_MEDIA_BACK 0xF1

4.6.32.5.186 HID_MEDIA_CALC
#define HID_MEDIA_CALC 0xFB

4.6.32.5.187 HID_MEDIA_COFFEE
#define HID_MEDIA_COFFEE 0xF9

4.6.32.5.188 HID_MEDIA_EDIT
#define HID_MEDIA_EDIT 0xF7

4.6.32.5.189 HID_MEDIA_EJECTCD
#define HID_MEDIA_EJECTCD 0xEC

4.6.32.5.190 HID_MEDIA_FIND
#define HID_MEDIA_FIND 0xF4

4.6.32.5.191 HID_MEDIA_FORWARD
#define HID_MEDIA_FORWARD 0xF2

4.6.32.5.192 HID_MEDIA_MUTE
#define HID_MEDIA_MUTE 0xEF

4.6.32.5.193 HID_MEDIA_NEXTSONG
#define HID_MEDIA_NEXTSONG 0xEB

4.6.32.5.194 HID_MEDIA_PLAYPAUSE
#define HID_MEDIA_PLAYPAUSE 0xE8

4.6.32.5.195 HID_MEDIA_PREVIOUSSONG
#define HID_MEDIA_PREVIOUSSONG 0xEA

4.6.32.5.196 HID_MEDIA_REFRESH
#define HID_MEDIA_REFRESH 0xFA

4.6.32.5.197 HID_MEDIA_SCROLLDOWN
#define HID_MEDIA_SCROLLDOWN 0xF6

4.6.32.5.198 HID_MEDIA_SCROLLUP
#define HID_MEDIA_SCROLLUP 0xF5

4.6.32.5.199 HID_MEDIA_SLEEP
#define HID_MEDIA_SLEEP 0xF8

4.6.32.5.200 HID_MEDIA_STOP
#define HID_MEDIA_STOP 0xF3

4.6.32.5.201 HID_MEDIA_STOPCD
#define HID_MEDIA_STOPCD 0xE9

4.6.32.5.202 HID_MEDIA_VOLUMEDOWN
#define HID_MEDIA_VOLUMEDOWN 0xEE

4.6.32.5.203 HID_MEDIA_VOLUMEUP
#define HID_MEDIA_VOLUMEUP 0xED

4.6.32.5.204 HID_MEDIA_WWW
#define HID_MEDIA_WWW 0xF0

4.6.32.5.205 HID_MENU
#define HID_MENU 0x76

4.6.32.5.206 HID_MODIFIER_LEFT_ALT
#define HID_MODIFIER_LEFT_ALT 0x04

4.6.32.5.207 HID_MODIFIER_LEFT_CTRL
#define HID_MODIFIER_LEFT_CTRL 0x01

4.6.32.5.208 HID_MODIFIER_LEFT_SHIFT
#define HID_MODIFIER_LEFT_SHIFT 0x02

4.6.32.5.209 HID_MODIFIER_LEFT_UI
#define HID_MODIFIER_LEFT_UI 0x08

4.6.32.5.210 HID_MODIFIER_NONE
#define HID_MODIFIER_NONE 0x00

4.6.32.5.211 HID_MODIFIER_RIGHT_ALT
#define HID_MODIFIER_RIGHT_ALT 0x40

4.6.32.5.212 HID_MODIFIER_RIGHT_CTRL
#define HID_MODIFIER_RIGHT_CTRL 0x10

4.6.32.5.213 HID_MODIFIER_RIGHT_SHIFT
#define HID_MODIFIER_RIGHT_SHIFT 0x20

4.6.32.5.214 HID_MODIFIER_RIGHT_UI
#define HID_MODIFIER_RIGHT_UI 0x80

4.6.32.5.215 HID_MUTE
#define HID_MUTE 0x7F

4.6.32.5.216 HID_N
#define HID_N 0x11

4.6.32.5.217 HID_O
#define HID_O 0x12

4.6.32.5.218 HID_OPEN_BRACE
#define HID_OPEN_BRACE 0x2F

4.6.32.5.219 HID_P
#define HID_P 0x13

4.6.32.5.220 HID_PAGEDOWN
#define HID_PAGEDOWN 0x4E

4.6.32.5.221 HID_PAGEUP
#define HID_PAGEUP 0x4B

4.6.32.5.222 HID_PASTE
#define HID_PASTE 0x7D

4.6.32.5.223 HID_PAUSE
#define HID_PAUSE 0x48

4.6.32.5.224 HID_POST_FAIL
#define HID_POST_FAIL 0x02

4.6.32.5.225 HID_POWER
#define HID_POWER 0x66

4.6.32.5.226 HID_PRINTSCREEN
#define HID_PRINTSCREEN 0x46

4.6.32.5.227 HID_Q
#define HID_Q 0x14

4.6.32.5.228 HID_R
#define HID_R 0x15

4.6.32.5.229 HID_RETURN
#define HID_RETURN 0x28

4.6.32.5.230 HID_RIGHT
#define HID_RIGHT 0x4F

4.6.32.5.231 HID_RIGHT_ALT
#define HID_RIGHT_ALT 0xE6

4.6.32.5.232 HID_RIGHT_CTRL
#define HID_RIGHT_CTRL 0xE4

4.6.32.5.233 HID_RIGHT_GUI
#define HID_RIGHT_GUI 0xE7

4.6.32.5.234 HID_RIGHT_SHIFT
#define HID_RIGHT_SHIFT 0xE5

4.6.32.5.235 HID_S
#define HID_S 0x16

4.6.32.5.236 HID_SCROLL_LOCK
#define HID_SCROLL_LOCK 0x47

4.6.32.5.237 HID_SELECT
#define HID_SELECT 0x77

4.6.32.5.238 HID_SEMICOLON
#define HID_SEMICOLON 0x33

4.6.32.5.239 HID_SLASH
#define HID_SLASH 0x38

4.6.32.5.240 HID_SPACEBAR
#define HID_SPACEBAR 0x2C

4.6.32.5.241 HID_STOP
#define HID_STOP 0x78

4.6.32.5.242 HID_T
#define HID_T 0x17

4.6.32.5.243 HID_TAB
#define HID_TAB 0x2B

4.6.32.5.244 HID_U
#define HID_U 0x18

4.6.32.5.245 HID_UNDERSCORE
#define HID_UNDERSCORE 0x2D

4.6.32.5.246 HID_UNDO

#define HID_UNDO 0x7A

4.6.32.5.247 HID_UP

#define HID_UP 0x52

4.6.32.5.248 HID_V

#define HID_V 0x19

4.6.32.5.249 HID_VOLUME_DOWN

#define HID_VOLUME_DOWN 0x81

4.6.32.5.250 HID_VOLUME_UP

#define HID_VOLUME_UP 0x80

4.6.32.5.251 HID_W

#define HID_W 0x1A

4.6.32.5.252 HID_X

#define HID_X 0x1B

4.6.32.5.253 HID_Y

#define HID_Y 0x1C

4.6.32.5.254 HID_Z

#define HID_Z 0x1D

4.6.33 source/source-files/usb_hid_mouse.c File Reference

Contains the implementation for the USB Mouse drivers.

```
#include <usb_hid_mouse.h>
#include <stddef.h>
#include <usb_common_elements.h>
#include <usb_hid.h>
#include <usb_hid_transfer.h>
#include <usb_core.h>
#include <usb_config.h>
#include <usb_protocol_hid.h>
```

4.6.33.1 Functions

- void USB_HIDMouseInitialize (USB_HID_REPORT_DESCRIPTOR_t *reportPtr)
  Sets up the mouse application for use with the HID class.
- RETURN_CODE_t USB_HIDMouseMove (int8_t x_position, int8_t y_position)
  Registers mouse movement and sends its coordinates to the host.
- RETURN_CODE_t USB_HIDMouseButton (bool buttonState, uint8_t button)
  Registers the button and its state and sends it to the host.
- RETURN_CODE_t USB_HIDMouseButtonLeft (bool buttonState)
  Registers the button state of the left button.
- RETURN_CODE_t USB_HIDMouseButtonRight (bool buttonState)
  Registers the button state of the right button.
- RETURN_CODE_t USB_HIDMouseButtonMiddle (bool buttonState)
  Registers the button state of the middle button.

4.6.33.2 Variables

- STATIC uint8_t usbHIDMouseRate

- STATIC uint8_t usbHIDMouseProtocol
- STATIC USB_MOUSE_REPORT_DATA_t mouseInputReport

4.6.33.3 Detailed Description

Contains the implementation for the USB Mouse drivers.

USBHIDMOUSE HID Mouse Source File

Version: USB Device Stack Driver Version 1.0.0

4.6.33.4 Variable Documentation

4.6.33.4.1 mouseInputReport

STATIC USB_MOUSE_REPORT_DATA_t mouseInputReport

4.6.33.4.2 usbHIDMouseProtocol

STATIC uint8_t usbHIDMouseProtocol

4.6.33.4.3 usbHIDMouseRate

STATIC uint8_t usbHIDMouseRate

4.6.34 source/source-files/usb_hid_mouse.h File Reference

```
#include <usb_common_elements.h>
#include <usb_protocol_hid.h>
#include <stdbool.h>
```

4.6.34.1 Functions

- void USB_HIDMouseInitialize (USB_HID_REPORT_DESCRIPTOR_t *reportPtr)
  Sets up the mouse application for use with the HID class.
- RETURN_CODE_t USB_HIDMouseMove (int8_t x_position, int8_t y_position)
  Registers mouse movement and sends its coordinates to the host.
- RETURN_CODE_t USB_HIDMouseButton (bool buttonState, uint8_t button)
  Registers the button and its state and sends it to the host.
- RETURN_CODE_t USB_HIDMouseButtonLeft (bool buttonState)
  Registers the button state of the left button.
- RETURN_CODE_t USB_HIDMouseButtonRight (bool buttonState)
  Registers the button state of the right button.
- RETURN_CODE_t USB_HIDMouseButtonMiddle (bool buttonState)
  Registers the button state of the middle button.

4.6.34.2 Macros

- #define USB_HID_MOUSE_REPORT_SIZE 8
  Size of the report for a standard HID mouse.

4.6.34.3 HID Mouse button state

Macros to signal the button state.

- #define HID_MOUSE_BUTTON_DOWN true
- #define HID_MOUSE_BUTTON_UP false

4.6.34.4 HID Mouse buttons

Macros for the mouse buttons.

- #define HID_MOUSE_LEFT_BUTTON 0x01u

- #define HID_MOUSE_RIGHT_BUTTON 0x02u
- #define HID_MOUSE_MIDDLE_BUTTON 0x04u

4.6.34.5 Detailed Description

USBHIDMOUSE HID Mouse Header File

4.6.35 source/source-files/usb_hid_transfer.c File Reference

Contains the implementation for the USB HID Transfer drivers.

```
#include <usb_hid_transfer.h>
#include <stddef.h>
#include <string.h>
#include <usb_common_elements.h>
#include <usb_hid.h>
#include <usb_core.h>
#include <usb_core_events.h>
#include <usb_core_transfer.h>
#include <usb_peripheral_read_write.h>
#include <usb_protocol_hid.h>
#include <usb_config.h>
```

4.6.35.1 Functions

- RETURN_CODE_t USB_HIDKeyboardReportInSend (USB_KEYBOARD_REPORT_DATA_t *data)
  Sends a HID keyboard input report to the interrupt IN endpoint.
- void USB_HIDKeyboardInputReportSentCallback (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)
  Verifies if a transfer was completed.
- RETURN_CODE_t USB_HIDMouseReportInSend (USB_MOUSE_REPORT_DATA_t *data)
  Sends a HID mouse input report to the interrupt IN endpoint.

4.6.35.2 Variables

- STATIC USB_PIPE_t keyboardPipe = {.address = USB_HID_INTERRUPT_EP, .direction = USB_EP_DIR_IN}
- STATIC USB_KEYBOARD_REPORT_DATA_t * nextKeyboardReport = NULL
- STATIC USB_KEYBOARD_REPORT_DATA_t keyboardReportBuffer
- STATIC USB_PIPE_t mousePipe = {.address = USB_HID_INTERRUPT_EP, .direction = USB_EP_DIR_IN}
- STATIC USB_MOUSE_REPORT_DATA_t mouseReportBuffer

4.6.35.3 Detailed Description

Contains the implementation for the USB HID Transfer drivers.

USBHIDTRANSFER HID Transfer Source File

Version: USB Device Stack HID Driver Version 1.0.0

4.6.35.4 Variable Documentation

4.6.35.4.1 keyboardPipe

STATIC USB_PIPE_t keyboardPipe = {.address = USB_HID_INTERRUPT_EP, .direction = USB_EP_DIR_IN}

4.6.35.4.2 keyboardReportBuffer

STATIC USB_KEYBOARD_REPORT_DATA_t keyboardReportBuffer

4.6.35.4.3 mousePipe

STATIC USB_PIPE_t mousePipe = {.address = USB_HID_INTERRUPT_EP, .direction = USB_EP_DIR_IN}

4.6.35.4.4 mouseReportBuffer

STATIC USB_MOUSE_REPORT_DATA_t mouseReportBuffer

4.6.35.4.5 nextKeyboardReport

STATIC USB_KEYBOARD_REPORT_DATA_t* nextKeyboardReport = NULL

4.6.36 source/source-files/usb_hid_transfer.h File Reference

```
#include <usb_common_elements.h>
#include <usb_protocol_hid.h>
#include <stdbool.h>
```

4.6.36.1 Functions

- RETURN_CODE_t USB_HIDKeyboardReportInSend (USB_KEYBOARD_REPORT_DATA_t *data)
  Sends a HID keyboard input report to the interrupt IN endpoint.
- void USB_HIDKeyboardInputReportSentCallback (USB_PIPE_t pipe, USB_TRANSFER_STATUS_t status, uint16_t bytesTransferred)
  Verifies if a transfer was completed.
- RETURN_CODE_t USB_HIDMouseReportInSend (USB_MOUSE_REPORT_DATA_t *data)
  Sends a HID mouse input report to the interrupt IN endpoint.

4.6.36.2 Detailed Description

USBHIDKTRANSFER HID Transfer Header File

4.6.37 source/source-files/usb_peripheral/usb_peripheral.c File Reference

Interface for a usb_peripheral module that needs to be implemented by a device specific USB module driver.

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include <usb_common_elements.h>
#include <usb_config.h>
#include <usb_peripheral.h>
#include <usb_peripheral_avr_du.h>
#include <usb_protocol_headers.h>
```

4.6.37.1 Functions

- bool USB_SetupIsReceived (void)
  Detects if the Setup event was received.
- bool USB_EventSOFIsReceived (void)
  Detects if the Start-of-Frame (SOF) event was received.
- void USB_EventSOFClear (void)
  Clears the SOF event.
- bool USB_EventResetIsReceived (void)
  Detects if the Reset event was received.
- void USB_EventResetClear (void)
  Clears the Reset event.
- uint8_t USB_EventOverUnderflowIsReceived (void)
  Detects if an Overflow and/or Underflow event was received.
- uint8_t USB_ControlOverUnderflowIsReceived (void)
  Detects if an Overflow and/or Underflow event was received on the control endpoints.
- void USB_EventOverUnderflowClear (void)
  Clears the Over/Underflow event.

- bool USB_EventSuspendIsReceived (void)
  Detects if a Suspend event was received.
- void USB_EventSuspendClear (void)
  Clears the Suspend event.
- bool USB_EventResumeIsReceived (void)
  Detects if a Resume event was received.
- void USB_EventResumeClear (void)
  Clears the Resume event.
- bool USB_EventStalledIsReceived (void)
  Detects if a Stalled event was received.
- void USB_EventStalledClear (void)
  Clears the Stalled event.
- void USB_BusAttach (void)
  Attaches the device to the USB bus.
- void USB_BusDetach (void)
  Detaches the device from the USB bus.
- bool USB_IsBusAttached (void)
  Checks if the device is attached to the USB bus not.
- void USB_DeviceAddressConfigure (uint8_t deviceAddress)
  Sets the device address.
- uint16_t USB_FrameNumberGet (void)
  Gets the current frame number.
- RETURN_CODE_t USB_ControlEndpointsInit (void)
  Ensures correct control endpoint initialization.
- RETURN_CODE_t USB_ControlSetupReceived (void)
  Verifies the received control setup.
- RETURN_CODE_t USB_ControlTransactionComplete (USB_PIPE_t pipe)
  Handles completed transactions on the control endpoints. Checks and verifies data OUT, data IN, ZLP OUT and ZLP IN.
- RETURN_CODE_t USB_ControlTransferZLP (uint8_t direction)
  Sends ZLP OUT and ZLP IN transactions on the control endpoints.
- RETURN_CODE_t USB_ControlTransferReset (void)
  Ensures correct control transfer reset.
- RETURN_CODE_t USB_ControlTransferDataSet (uint8_t *dataPtr, uint16_t dataSize)
  Updates the transfer data pointer and size in ControlTransfer.
- RETURN_CODE_t USB_ControlTransferDataWriteBuffer (uint8_t *dataPtr, uint8_t dataSize)
  Copies data to the transfer buffer and sets the transfer data pointer and size in ControlTransfer.
- void USB_ControlEndOfRequestCallbackRegister (USB_SETUP_ENDOFREQUEST_CALLBACK_t callback)
  Sets the callback for end of a control request.
- void USB_ControlProcessSetupCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)
  Sets the callback for the setup processing.

- void USB_ControlOverUnderRunCallbackRegister (USB_SETUP_OVERUNDERRUN_CALLBACK_t callback)
  Sets the callback for a control overrun or underrun.
- RETURN_CODE_t USB_ControlProcessOverUnderflow (uint8_t overunderflow)
  Handles the control Over/Underflow events.
- RETURN_CODE_t USB_HandleEventStalled (USB_PIPE_t pipe)
  Handles the Stall events.
- void USB_PeripheralInitialize (void)
  Enables the peripheral and the frame number, enables and resets FIFO, sets the endpoint table address and max endpoints.
- void USB_PeripheralDisable (void)
  Disables the USB peripheral and aborts any ongoing transaction.

4.6.37.2 Variables

- STATIC USB_CONTROL_TRANSFER_t controlTransfer = { .transferDataPtr = controlTransfer.buffer }

4.6.37.3 Detailed Description

Interface for a usb_peripheral module that needs to be implemented by a device specific USB module driver.

USBPERIPHERAL Peripheral Source File

Version: USB Device Stack HAL Driver Version 1.0.0

4.6.37.4 Variable Documentation

4.6.37.4.1 controlTransfer

STATIC USB_CONTROL_TRANSFER_t controlTransfer = { .transferDataPtr = controlTransfer.buffer }

4.6.38 source/source-files/usb_peripheral/usb_peripheral.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include <usb_common_elements.h>
#include <usb_peripheral_endpoint.h>
#include <usb_peripheral_read_write.h>
#include <usb_protocol_headers.h>
```

4.6.38.1 Data structures

- struct USB_CONTROL_TRANSFER_struct

4.6.38.2 Functions

- bool USB_SetupIsReceived (void)
  Detects if the Setup event was received.
- bool USB_EventSOFIsReceived (void)
  Detects if the Start-of-Frame (SOF) event was received.
- void USB_EventSOFClear (void)
  Clears the SOF event.
- bool USB_EventResetIsReceived (void)
  Detects if the Reset event was received.
- void USB_EventResetClear (void)
  Clears the Reset event.
- uint8_t USB_EventOverUnderflowIsReceived (void)

Detects if an Overflow and/or Underflow event was received.

- uint8_t USB_ControlOverUnderflowIsReceived (void)
  Detects if an Overflow and/or Underflow event was received on the control endpoints.
- void USB_EventOverUnderflowClear (void)
  Clears the Over/Underflow event.
- bool USB_EventSuspendIsReceived (void)
  Detects if a Suspend event was received.
- void USB_EventSuspendClear (void)
  Clears the Suspend event.
- bool USB_EventResumeIsReceived (void)
  Detects if a Resume event was received.
- void USB_EventResumeClear (void)
  Clears the Resume event.
- bool USB_EventStalledIsReceived (void)
  Detects if a Stalled event was received.
- void USB_EventStalledClear (void)
  Clears the Stalled event.
- void USB_BusAttach (void)
  Attaches the device to the USB bus.
- void USB_BusDetach (void)
  Detaches the device from the USB bus.
- bool USB_IsBusAttached (void)
  Checks if the device is attached to the USB bus not.
- void USB_DeviceAddressConfigure (uint8_t deviceAddress)
  Sets the device address.
- uint16_t USB_FrameNumberGet (void)
  Gets the current frame number.
- RETURN_CODE_t USB_ControlEndpointsInit (void)
  Ensures correct control endpoint initialization.
- RETURN_CODE_t USB_ControlSetupReceived (void)
  Verifies the received control setup.
- RETURN_CODE_t USB_ControlTransactionComplete (USB_PIPE_t pipe)
  Handles completed transactions on the control endpoints. Checks and verifies data OUT, data IN, ZLP OUT and ZLP IN.
- RETURN_CODE_t USB_ControlTransferZLP (uint8_t direction)
  Sends ZLP OUT and ZLP IN transactions on the control endpoints.
- RETURN_CODE_t USB_ControlTransferReset (void)
  Ensures correct control transfer reset.
- RETURN_CODE_t USB_ControlTransferDataSet (uint8_t *dataPtr, uint16_t dataSize)
  Updates the transfer data pointer and size in ControlTransfer.
- RETURN_CODE_t USB_ControlTransferDataWriteBuffer (uint8_t *dataPtr, uint8_t dataSize)
  Copies data to the transfer buffer and sets the transfer data pointer and size in ControlTransfer.

- void USB_ControlEndOfRequestCallbackRegister (USB_SETUP_ENDOFREQUEST_CALLBACK_t callback)
  Sets the callback for end of a control request.
- void USB_ControlProcessSetupCallbackRegister (USB_SETUP_PROCESS_CALLBACK_t callback)
  Sets the callback for the setup processing.
- void USB_ControlOverUnderRunCallbackRegister (USB_SETUP_OVERUNDERRUN_CALLBACK_t callback)
  Sets the callback for a control overrun or underrun.
- RETURN_CODE_t USB_ControlProcessOverUnderflow (uint8_t overunderflow)
  Handles the control Over/Underflow events.
- RETURN_CODE_t USB_HandleEventStalled (USB_PIPE_t pipe)
  Handles the Stall events.
- void USB_PeripheralInitialize (void)
  Enables the peripheral and the frame number, enables and resets FIFO, sets the endpoint table address and max endpoints.
- void USB_PeripheralDisable (void)
  Disables the USB peripheral and aborts any ongoing transaction.

4.6.38.3 Typedefs

- typedef struct USB_CONTROL_TRANSFER_struct USB_CONTROL_TRANSFER_t

4.6.38.4 Detailed Description

USBPERIPHERAL Peripheral Header File

4.6.38.5 Typedef Documentation

4.6.38.5.1 USB_CONTROL_TRANSFER_t

typedef struct USB_CONTROL_TRANSFER_struct USB_CONTROL_TRANSFER_t

4.6.39 source/source-files/usb_peripheral/usb_peripheral_avr_du.h File Reference

```
#include <avr/io.h>
#include <usb_config.h>
#include <usb_protocol_headers.h>
```

4.6.39.1 Data structures

- struct USB_ENDPOINT_TABLE_struct
  Represents the endpoint configuration table based on the number of endpoints in use. The table data structure is defined by USB_EP_TABLE_struct in the device header file, modified to support configuration of size from USB_EP_NUM.
- struct USB_PIPE_TRANSFER_struct
  Represents a transfer created for a pipe, either IN or OUT.

4.6.39.2 Functions

- static ALWAYS_INLINE void WaitUntilRMWDone (void)
  Waits until a Read-Modify-Write operation is done. This blocking wait operation is expected to complete within 14 clock cycles.
- static ALWAYS_INLINE void USB_EndPointOutDisable (uint8_t endpointAddress)
  Disables the OUT endpoint with the given address.
- static ALWAYS_INLINE void USB_EndPointInDisable (uint8_t endpointAddress)
  Disables the IN endpoint with the given address.

- static ALWAYS_INLINE bool USB_EndPointOutIsEnabled (uint8_t endpointAddress)
  Checks if the OUT endpoint at the given address is enabled.
- static ALWAYS_INLINE bool USB_EndPointInIsEnabled (uint8_t endpointAddress)
  Checks if the IN endpoint at the given address is enabled.
- static ALWAYS_INLINE uint8_t USB_EndPointOutTypeConfigGet (uint8_t endpointAddress)
  Gets the OUT endpoint configuration at the given address.
- static ALWAYS_INLINE uint8_t USB_EndPointInTypeConfigGet (uint8_t endpointAddress)
  Gets the IN endpoint configuration at the given address.
- static ALWAYS_INLINE void USB_EndpointOutControlSet (uint8_t endpointAddress, uint8_t value)
  Sets endpoint control OUT.
- static ALWAYS_INLINE void USB_EndpointInControlSet (uint8_t endpointAddress, uint8_t value)
  Sets endpoint control IN.
- static ALWAYS_INLINE void USB_EndpointOutStatusClear (uint8_t endpointAddress)
  Clears OUT endpoint status.
- static ALWAYS_INLINE void USB_EndpointInStatusClear (uint8_t endpointAddress)
  Clears IN endpoint status.
- static ALWAYS_INLINE void USB_EndpointOutDefaultSizeSet (uint8_t endpointAddress, uint8_t endpointSizeConfig)
  Sets the endpoint size for a default type OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInDefaultSizeSet (uint8_t endpointAddress, uint8_t endpointSizeConfig)
  Sets the endpoint size for a default type IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutIsoSizeSet (uint8_t endpointAddress, uint8_t endpointSizeConfig)
  Sets the endpoint size for an isochronous OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInIsoSizeSet (uint8_t endpointAddress, uint8_t endpointSizeConfig)
  Sets the endpoint size for an isochronous IN endpoint.
- static ALWAYS_INLINE uint8_t USB_EndpointOutDefaultSizeGet (uint8_t endpointAddress)
  Gets the size of a default type OUT endpoint.
- static ALWAYS_INLINE uint8_t USB_EndpointInDefaultSizeGet (uint8_t endpointAddress)
  Gets the size of a default type IN endpoint.
- static ALWAYS_INLINE uint8_t USB_EndpointOutIsoSizeGet (uint8_t endpointAddress)
  Gets the size of an isochronous OUT endpoint.
- static ALWAYS_INLINE uint8_t USB_EndpointInIsoSizeGet (uint8_t endpointAddress)
  Gets the size of an isochronous IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutTransactionCompleteInterruptEnable (uint8_t endpointAddress)
  Enables transaction complete interrupt for the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInTransactionCompleteInterruptEnable (uint8_t endpointAddress)
  Enables transaction complete interrupt for the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutTransactionCompleteInterruptDisable (uint8_t endpointAddress)

Disables transaction complete interrupt for the specified OUT endpoint.

- static ALWAYS_INLINE void USB_EndpointInTransactionCompleteDisable (uint8_t endpointAddress)
  Disables transaction complete interrupt for the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutMultipktEnable (uint8_t endpointAddress)
  Enables multipacket for the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInMultipktEnable (uint8_t endpointAddress)
  Enables multipacket for the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutMultipktDisable (uint8_t endpointAddress)
  Disables multipacket for the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInMultipktDisable (uint8_t endpointAddress)
  Disables multipacket for the specified IN endpoint.
- static ALWAYS_INLINE bool USB_EndpointOutMultipktIsEnabled (uint8_t endpointAddress)
  Checks if multipacket is enabled on the specified OUT endpoint.
- static ALWAYS_INLINE bool USB_EndpointInMultipktIsEnabled (uint8_t endpointAddress)
  Checks if multipacket is enabled on the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutAzlpEnable (uint8_t endpointAddress)
  Enables Auto Zero Length Packet (AZLP) on the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInAlzpEnable (uint8_t endpointAddress)
  Enables AZLP on the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutAzlpDisable (uint8_t endpointAddress)
  Disables AZLP on the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInAzlpDisable (uint8_t endpointAddress)
  Disables AZLP on the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutStall (uint8_t endpointAddress)
  Stalls the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInStall (uint8_t endpointAddress)
  Stalls the specified IN endpoint.
- static ALWAYS_INLINE void USB_EndpointOutStallClear (uint8_t endpointAddress)
  Stops stalling the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInStallClear (uint8_t endpointAddress)
  Stops stalling the specified IN endpoint.
- static ALWAYS_INLINE bool USB_EndpointOutIsStalled (uint8_t endpointAddress)
  Checks if the specified OUT endpoint is stalled.
- static ALWAYS_INLINE bool USB_EndpointInIsStalled (uint8_t endpointAddress)
  Checks if the specified IN endpoint is stalled.
- static ALWAYS_INLINE void USB_EndpointOutStallAck (uint8_t endpointAddress)
  Acknowledges that an OUT endpoint is stalled and Clears the USB STATUS register.
- static ALWAYS_INLINE void USB_EndpointInStallAck (uint8_t endpointAddress)
  Acknowledges that an IN endpoint is stalled and Clears the USB STATUS register.
- static ALWAYS_INLINE void USB_EndpointOutNAKSet (uint8_t endpointAddress)
  Sets OUT endpoint status to NAK.

- static ALWAYS_INLINE void USB_EndpointInNAKSet (uint8_t endpointAddress)
  Sets IN endpoint status to NAK.
- static ALWAYS_INLINE void USB_EndpointOutNAKClear (uint8_t endpointAddress)
  Clears the USB NAK status from the OUT endpoint STATUS register.
- static ALWAYS_INLINE void USB_EndpointInNAKClear (uint8_t endpointAddress)
  Clears the USB NAK status from the IN endpoint STATUS register.
- static ALWAYS_INLINE bool USB_EndpointOutNAKIsSet (uint8_t endpointAddress)
  Checks the OUT endpoint STATUS register for the NAK status.
- static ALWAYS_INLINE bool USB_EndpointInNAKIsSet (uint8_t endpointAddress)
  Checks the OUT endpoint STATUS register for the NAK status.
- static ALWAYS_INLINE void USB_EndpointOutTransactionCompleteAck (uint8_t endpointAddress)
  Acknowledges the transaction complete status on a specified OUT endpoint and Clears the USB STATUS register.
- static ALWAYS_INLINE void USB_EndpointInTransactionCompleteAck (uint8_t endpointAddress)
  Acknowledges the transaction complete status on a specified IN endpoint and Clears the USB STATUS register.
- static ALWAYS_INLINE bool USB_EndpointOutTransactionIsComplete (uint8_t endpointAddress)
  Checks if the USB OUT endpoint has the Transaction Complete status.
- static ALWAYS_INLINE bool USB_EndpointInTransactionIsComplete (uint8_t endpointAddress)
  Checks if the USB IN endpoint has the Transaction Complete status.
- static ALWAYS_INLINE void USB_EndpointOutSetupReceivedAck (uint8_t endpointAddress)
  Acknowledges the Setup Received status on a specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInSetupCompleteAck (uint8_t endpointAddress)
  Acknowledges the Setup Received status on a specified IN endpoint.
- static ALWAYS_INLINE bool USB_EndpointOutSetupIsReceived (uint8_t endpointAddress)
  Checks if the USB OUT endpoint has the Setup Received status.
- static ALWAYS_INLINE bool USB_EndpointInSetupIsReceived (uint8_t endpointAddress)
  Checks if the USB IN endpoint has the Setup Received status.
- static ALWAYS_INLINE void USB_EndpointOutDataToggleSet (uint8_t endpointAddress)
  Sets OUT endpoint data toggle.
- static ALWAYS_INLINE void USB_EndpointInDataToggleSet (uint8_t endpointAddress)
  Sets IN endpoint data toggle.
- static ALWAYS_INLINE void USB_EndpointOutDataToggleClear (uint8_t endpointAddress)
  Clears OUT endpoint data toggle.
- static ALWAYS_INLINE void USB_EndpointInDataToggleClear (uint8_t endpointAddress)
  Clears IN endpoint data toggle.
- static ALWAYS_INLINE bool USB_EndpointOutDataToggleIsSet (uint8_t endpointAddress)
  Checks if data toggle is set on the specified OUT endpoint.
- static ALWAYS_INLINE bool USB_EndpointInDataToggleIsSet (uint8_t endpointAddress)
  Checks if data toggle is set on the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointOutBufferSet (uint8_t endpointAddress, uint8_t *bufAddress)
  Sets endpoint buffer OUT.

- static ALWAYS_INLINE void USB_EndpointInBufferSet (uint8_t endpointAddress, uint8_t *bufAddress)
  Sets endpoint buffer IN.
- static ALWAYS_INLINE void USB_NumberBytesToSendSet (uint8_t endpointAddress, uint16_t numberBytes)
  Sets how many bytes of data are intended to be sent from the specified endpoint.
- static ALWAYS_INLINE uint16_t USB_NumberBytesToSendGet (uint8_t endpointAddress)
  Reads out the CNT register to know how many bytes of data are intended to be sent from the specified endpoint.
- static ALWAYS_INLINE void USB_NumberBytesToSendReset (uint8_t endpointAddress)
  Clears the CNT register to tell the peripheral no data is intended to be sent from the specified endpoint.
- static ALWAYS_INLINE uint16_t USB_NumberBytesSentGet (uint8_t endpointAddress)
  Reads out how many bytes have been sent from the specified endpoint.
- static ALWAYS_INLINE void USB_NumberBytesSentReset (uint8_t endpointAddress)
  Clears the MCNT register that keeps track of how many bytes of data have been sent.
- static ALWAYS_INLINE void USB_NumberBytesToReceiveSet (uint8_t endpointAddress, uint16_t numberBytes)
  Sets how many bytes of data are expected to be received on a specified endpoint.
- static ALWAYS_INLINE uint16_t USB_NumberBytesToReceiveGet (uint8_t endpointAddress)
  Gets how many bytes of data are expected to be received on a specified endpoint.
- static ALWAYS_INLINE void USB_NumberBytesToReceiveReset (uint8_t endpointAddress)
  Clears the MCNT register to tell the peripheral no data is intended to be received on the specified endpoint.
- static ALWAYS_INLINE uint16_t USB_NumberBytesReceivedGet (uint8_t endpointAddress)
  Gets how many bytes of data have been received on a specified endpoint.
- static ALWAYS_INLINE void USB_NumberBytesReceivedReset (uint8_t endpointAddress)
  Resets the counter that counts amount of bytes of data received on a specific endpoint.
- static ALWAYS_INLINE bool USB_EndpointOutOverUnderflowIsSet (uint8_t endpointAddress)
  Checks if OUT endpoint has overflowed.
- static ALWAYS_INLINE bool USB_EndpointInOverUnderflowIsSet (uint8_t endpointAddress)
  Checks if IN endpoint has underflowed.
- static ALWAYS_INLINE void USB_EndpointOutOverUnderflowAck (uint8_t endpointAddress)
  Acknowledges overflow on the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInOverUnderflowAck (uint8_t endpointAddress)
  Acknowledges underflow on the specified IN endpoint.
- static ALWAYS_INLINE bool USB_EndpointOutCRCHasFailed (uint8_t endpointAddress)
  Checks if the specified OUT endpoint has a Cyclic Redundancy Check (CRC) failure.
- static ALWAYS_INLINE bool USB_EndpointInCRCHasFailed (uint8_t endpointAddress)
  Checks if the specified IN endpoint has a CRC failure.
- static ALWAYS_INLINE void USB_EndpointOutCRCAck (uint8_t endpointAddress)
  Acknowledges a CRC failure on the specified OUT endpoint.
- static ALWAYS_INLINE void USB_EndpointInCRCAck (uint8_t endpointAddress)
  Acknowledges a CRC failure on the specified IN endpoint.

- static ALWAYS_INLINE void USB_GlobalNAKEnable (void)
  Enables global NAK.
- static ALWAYS_INLINE void USB_GlobalNAKDisable (void)
  Disables global NAK.
- static ALWAYS_INLINE bool USB_GlobalNAKIsEnable (void)
  Checks the global NAK setting.
- static ALWAYS_INLINE void USB_ConnectionAttach (void)
  Tells the USB peripheral to attach.
- static ALWAYS_INLINE void USB_ConnectionDetach (void)
  Tells the USB peripheral to detach.
- static ALWAYS_INLINE bool USB_ConnectionIsAttach (void)
  Checks if the USB connection is attached.
- static ALWAYS_INLINE void USB_Enable (void)
  Enables the USB peripheral.
- static ALWAYS_INLINE void USB_Disable (void)
  Disables the USB peripheral.
- static ALWAYS_INLINE bool USB_IsEnable (void)
  Checks if the USB peripheral is enabled.
- static ALWAYS_INLINE void USB_FifoEnable (void)
  Enables USB FIFO.
- static ALWAYS_INLINE void USB_FifoDisable (void)
  Disables USB FIFO.
- static ALWAYS_INLINE bool USB_FifoIsEnable (void)
  Checks if USB FIFO has been enabled.
- static ALWAYS_INLINE void USB_AutomaticGlobalNAKEnable (void)
  Enables automatic global NAK for the USB peripheral.
- static ALWAYS_INLINE void USB_AutomaticGlobalNAKDisable (void)
  Disables automatic global NAK for the USB peripheral.
- static ALWAYS_INLINE bool USB_AutomaticGlobalNAKIsEnable (void)
  Checks if automatic global NAK has been enabled.
- static ALWAYS_INLINE void USB_FrameNumEnable (void)
  Enables storing the last SOF token frame number in FRAMENUM. This is a device-specific function.
- static ALWAYS_INLINE void USB_FrameNumDisable (void)
  Disables storing the last SOF token frame number in FRAMENUM. This is a device-specific function.
- static ALWAYS_INLINE bool USB_FrameNumIsEnable (void)
  Checks if storing of the last SOF token frame number is enabled. This is a device-specific function.
- static ALWAYS_INLINE uint16_t USB_FrameNumGet (void)
  Gets the current frame number.
- static ALWAYS_INLINE void USB_MaxEndpointsSet (uint8_t maxEndpoint)
  Sets maximum number of endpoint addresses used by the USB peripheral.
- static ALWAYS_INLINE void USB_MaxEndpointsReset (void)

Clears the USB endpoint maximum, setting the maximum endpoint to EP0.

- static ALWAYS_INLINE uint8_t USB_MaxEndpointsGet (void)
  Checks what the maximum number of endpoint addresses is.
- static ALWAYS_INLINE void USB_EndpointTableAddressSet (USB_EP_PAIR_t *endpointTableAddress)
  Sets the address of the endpoint table. This is a device-specific function.
- static ALWAYS_INLINE void USB_EndpointTableAddressReset (void)
  Sets the address of the endpoint table to 0. This is a device-specific function.
- static ALWAYS_INLINE uint16_t USB_EndpointTableAddressGet (void)
  Gets the address of the endpoint table. This is a device-specific function.
- static ALWAYS_INLINE void USB_FifoReadPointerReset (void)
  Resets the read FIFO pointer. This is a device-specific function.
- static ALWAYS_INLINE int8_t USB_FifoReadPointerGet (void)
  Gets the read FIFO pointer. This is a device-specific function.
- static ALWAYS_INLINE void USB_FifoWritePointerReset (void)
  Resets the write FIFO pointer. This is a device-specific function.
- static ALWAYS_INLINE int8_t USB_FifoWritePointerGet (void)
  Gets the write FIFO pointer. This is a device-specific function.
- static ALWAYS_INLINE void USB_DeviceAddressSet (uint8_t usbAddress)
  Sets the device address.
- static ALWAYS_INLINE void USB_DeviceAddressReset (void)
  Resets the device address.
- static ALWAYS_INLINE uint8_t USB_DeviceAddressGet (void)
  Gets the device address.
- static ALWAYS_INLINE void USB_UpstreamResumeEnable (void)
  Enables an upstream resume to be initated.
- static ALWAYS_INLINE bool USB_UpstreamResumeIsEnable (void)
  Checks if upstream resume is enabled, but not yet initiated.
- static ALWAYS_INLINE uint8_t USB_BusStateGet (void)
  Gets the USB bus state.
- static ALWAYS_INLINE bool USB_BusStateIs (uint8_t bus_state_bm)
  Checks if the USB bus has any specific status flags set.
- static ALWAYS_INLINE void USB_SOFInterruptEnable (void)
  Enables the USB Start-Of-Frame interrupt.
- static ALWAYS_INLINE void USB_SOFInterruptDisable (void)
  Disables the USB Start-Of-Frame interrupt.
- static ALWAYS_INLINE void USB_SOFInterruptClear (void)
  Clears the USB Start-Of-Frame Interrupt flag.
- static ALWAYS_INLINE bool USB_SOFInterruptIs (void)
  Checks if the USB Start-Of-Frame interrupt has been triggered.
- static ALWAYS_INLINE void USB_SuspendInterruptEnable (void)
  Enables the USB Suspend interrupt.

- static ALWAYS_INLINE void USB_SuspendInterruptDisable (void)
  Disables the USB Suspend interrupt.
- static ALWAYS_INLINE void USB_SuspendInterruptClear (void)
  Clears the USB Suspend Interrupt flag.
- static ALWAYS_INLINE bool USB_SuspendInterruptIs (void)
  Checks if the USB Suspend interrupt has been triggered.
- static ALWAYS_INLINE void USB_ResumeInterruptEnable (void)
  Enables the USB Resume interrupt.
- static ALWAYS_INLINE void USB_ResumeInterruptDisable (void)
  Disables the USB Resume interrupt.
- static ALWAYS_INLINE void USB_ResumeInterruptClear (void)
  Clears the USB Resume Interrupt flag.
- static ALWAYS_INLINE bool USB_ResumeInterruptIs (void)
  Checks if the USB Resume interrupt has been triggered.
- static ALWAYS_INLINE void USB_ResetInterruptEnable (void)
  Enables the USB Reset interrupt.
- static ALWAYS_INLINE void USB_ResetInterruptDisable (void)
  Disables the USB Reset interrupt.
- static ALWAYS_INLINE void USB_ResetInterruptClear (void)
  Clears the USB Reset Interrupt flag.
- static ALWAYS_INLINE bool USB_ResetInterruptIs (void)
  Checks if the USB Reset interrupt has been triggered.
- static ALWAYS_INLINE void USB_StalledInterruptEnable (void)
  Enables the USB Stalled interrupt.
- static ALWAYS_INLINE void USB_StalledInterruptDisable (void)
  Disables the USB Stalled interrupt.
- static ALWAYS_INLINE void USB_StalledInterruptClear (void)
  Clears the USB Stalled Interrupt flag.
- static ALWAYS_INLINE bool USB_StalledInterruptIs (void)
  Checks if the USB Stalled interrupt has been triggered.
- static ALWAYS_INLINE void USB_UnderflowInterruptEnable (void)
  Enables the USB Underflow interrupt.
- static ALWAYS_INLINE void USB_UnderflowInterruptDisable (void)
  Disables the USB Underflow interrupt.
- static ALWAYS_INLINE void USB_UnderflowInterruptClear (void)
  Clears the USB Underflow Interrupt flag.
- static ALWAYS_INLINE bool USB_UnderflowInterruptIs (void)
  Checks if an Underflow interrupt has been triggered.
- static ALWAYS_INLINE void USB_OverflowInterruptEnable (void)
  Enables the USB Overflow interrupt.
- static ALWAYS_INLINE void USB_OverflowInterruptDisable (void)
  Disables the USB Overflow interrupt.

- static ALWAYS_INLINE void USB_OverflowInterruptClear (void)
  Clears the USB Overflow Interrupt flag.
- static ALWAYS_INLINE bool USB_OverflowInterruptIs (void)
  Checks if an Overflow interrupt has been triggered.
- static ALWAYS_INLINE void USB_TransactionCompleteInterruptEnable (void)
  Enables the USB Transaction Complete interrupt.
- static ALWAYS_INLINE void USB_TransactionCompleteInterruptDisable (void)
  Disables the USB Transaction Complete interrupt.
- static ALWAYS_INLINE void USB_TransactionCompleteInterruptAck (void)
  Clears the USB Transaction Complete Interrupt flag.
- static ALWAYS_INLINE bool USB_TransactionCompleteInterruptIs (void)
  Checks if a Transaction Complete interrupt has been triggered.
- static ALWAYS_INLINE bool USB_ReadModifyWriteInterruptIs (void)
  Checks if the USB Read-Modify-Write Interrupt is enabled.
- static ALWAYS_INLINE void USB_GlobalNAKDoneInterruptEnable (void)
  Enables the USB Global NAK Done interrupt.
- static ALWAYS_INLINE void USB_GlobalNAKDoneInterruptDisable (void)
  Disables the USB Global NAK Done interrupt.
- static ALWAYS_INLINE void USB_GlobalNAKDoneInterruptAck (void)
  Clears the USB Global NAK Done Interrupt flag.
- static ALWAYS_INLINE bool USB_GlobalNAKDoneInterruptIs (void)
  Checks if the USB Global NAK Done interrupt has been triggered.
- static ALWAYS_INLINE void USB_SetupInterruptEnable (void)
  Enables the USB Setup interrupt.
- static ALWAYS_INLINE void USB_SetupInterruptDisable (void)
  Disables the USB Setup interrupt.
- static ALWAYS_INLINE void USB_SetupInterruptClear (void)
  Clears the USB Setup Interrupt flag.
- static ALWAYS_INLINE bool USB_SetupInterruptIs (void)
  Checks if a USB Setup interrupt has been triggered.
- static ALWAYS_INLINE void USB_InterruptFlagsClear (void)
  Clears all the USB Interrupt flags.

4.6.39.3 Macros

- #define ALWAYS_INLINE __attribute__((always_inline)) inline
  Alias that makes always inline function definitions more readable.

4.6.39.4 Typedefs

- typedef struct USB_ENDPOINT_TABLE_struct USB_ENDPOINT_TABLE_t
- typedef struct USB_PIPE_TRANSFER_struct USB_PIPE_TRANSFER_t

4.6.39.5 Variables

- USB_ENDPOINT_TABLE_t endpointTable

SRAM tables for the FIFO and endpoint registers, as well as the FRAMENUM register. Represents the endpoint configuration table based on the number of endpoints in use. This line instantiates an object using the data structure type.

4.6.39.6 Detailed Description

USBPERIPHERALAVRDU Peripheral AVR DU Specific Header File

4.6.39.7 Typedef Documentation

4.6.39.7.1 USB_ENDPOINT_TABLE_t

typedef struct USB_ENDPOINT_TABLE_struct USB_ENDPOINT_TABLE_t

4.6.39.7.2 USB_PIPE_TRANSFER_t

typedef struct USB_PIPE_TRANSFER_struct USB_PIPE_TRANSFER_t

4.6.40 source/source-files/usb_peripheral/usb_peripheral_endpoint.c File Reference

API module for usb_peripheral covering endpoint related functions.

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include <usb_common_elements.h>
#include <usb_config.h>
#include <usb_peripheral_avr_du.h>
#include <usb_peripheral_endpoint.h>
#include <usb_protocol_headers.h>
```

4.6.40.1 Functions

- RETURN_CODE_t USB_EndpointConfigure (USB_PIPE_t pipe, uint16_t endpointSize, USB_ENDPOINT_t endpointType)

  Configures the endpoint with the desired settings using the Control and Status Register. Used to set up an endpoint before using it in an application. Sets up all the control register settings by looking up the usb_config.h file and clears the count registers.
- RETURN_CODE_t USB_EndpointDisable (USB_PIPE_t pipe)

  Disables the endpoint by setting the endpoint type to 0x00.
- uint16_t USB_EndpointSizeGet (USB_PIPE_t pipe)

  Helper function to return the endpoint size.
- USB_ENDPOINT_t USB_EndpointTypeGet (USB_PIPE_t pipe)

  Helper function to return the endpoint type.
- RETURN_CODE_t USB_EndpointStall (USB_PIPE_t pipe)

  Helps stall an endpoint when a command received from the host is invalid or unrecognizable. Used if the host sends data that is not supported by the device.
- RETURN_CODE_t USB_EndpointStallClear (USB_PIPE_t pipe)

  Helps clear the Stall condition after the device has recovered from an unsupported command from the host. Used to reset stall before the next USB transfer. Used when the host issues a clear HALT/Feature request to reset stall.
- bool USB_EndpointIsStalled (USB_PIPE_t pipe)

  Helper function to return the endpoint Stall condition.
- RETURN_CODE_t USB_EndpointStalledConditionAck (USB_PIPE_t pipe)

  Acknowledges the stall status condition by clearing the Stall Status bit. Used to clear the Stall Status bit after a stall has been detected.
- RETURN_CODE_t USB_DataToggleSet (USB_PIPE_t pipe)

Sets the Data Toggle bit on an endpoint which is used to ensure correct data sequence. Only used if hardware data toggling is not available. After a successful transaction, toggle the Data Toggle bit. For SETUP transactions, ensure that the SETUP stage clears the Data Toggle bit, while the data stage and status stage set the Data Toggle bit.

- RETURN_CODE_t USB_DataToggleClear (USB_PIPE_t pipe)

  Clears the Data Toggle bit on an endpoint which is used to ensure correct data sequence. Only used if hardware data toggling is not available. After a successful transaction, toggle the Data Toggle bit. For SETUP transactions, ensure that the SETUP stage clears the Data Toggle bit, while the data stage and status stage set the Data Toggle bit.
- RETURN_CODE_t USB_DataToggle (USB_PIPE_t pipe)

  Toggles the Data Toggle bit on an endpoint which is used to ensure correct data sequence. Only used if hardware data toggling is not available. After a successful transaction, toggle the Data Toggle bit. For SETUP transactions, ensure that the SETUP stage clears the Data Toggle bit, while the data stage and status stage set the Data Toggle bit.
- RETURN_CODE_t ConvertEndpointSizeToMask (uint16_t endpointSize, USB_ENDPOINT_t endpointType, uint8_t *endpointMaskPtr)

  Converts an endpoint size in number of bytes into a register setting. Converts the endpoint size bit mask based on the EP_BUFSIZE setting of the endpoint control register.
- RETURN_CODE_t EndpointBufferSet (USB_PIPE_t pipe, uint8_t *bufAddress)

  Configures the endpoint data buffer to a location in RAM for the next transaction.

4.6.40.2 Macros

- #define IsPowerOfTwo(number) ((0u != (number)) && (((number) & ((number)-1u)) == 0u))

  Algorithm to detect if a given number is a power of two. A number is a power of two if it has exactly one '1' in its binary representation. This is true if subtracting '1' from the number and doing an AND operation on the result with the number itself returns 0.

4.6.40.3 Variables

- USB_ENDPOINT_TABLE_t endpointTable

  SRAM tables for the FIFO and endpoint registers, as well as the FRAMENUM register. Represents the endpoint configuration table based on the number of endpoints in use. This line instantiates an object using the data structure type.

4.6.40.4 Detailed Description

API module for usb_peripheral covering endpoint related functions.

USBPERIPHERALENDPOINT Peripheral Endpoint Source File

Version: USB Device Stack HAL Driver Version 1.0.0

4.6.41 source/source-files/usb_peripheral/usb_peripheral_endpoint.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "usb_common_elements.h"
#include "usb_protocol_headers.h"
```

4.6.41.1 Functions

- RETURN_CODE_t USB_EndpointConfigure (USB_PIPE_t pipe, uint16_t endpointSize, USB_ENDPOINT_t endpointType)

  Configures the endpoint with the desired settings using the Control and Status Register. Used to set up an endpoint before using it in an application. Sets up all the control register settings by looking up the usb_config.h file and clears the count registers.
- RETURN_CODE_t USB_EndpointDisable (USB_PIPE_t pipe)

Disables the endpoint by setting the endpoint type to 0x00.

- uint16_t USB_EndpointSizeGet (USB_PIPE_t pipe)
  Helper function to return the endpoint size.
- USB_ENDPOINT_t USB_EndpointTypeGet (USB_PIPE_t pipe)
  Helper function to return the endpoint type.
- RETURN_CODE_t USB_EndpointStall (USB_PIPE_t pipe)
  Helps stall an endpoint when a command received from the host is invalid or unrecognizable. Used if the host sends data that is not supported by the device.
- RETURN_CODE_t USB_EndpointStallClear (USB_PIPE_t pipe)
  Helps clear the Stall condition after the device has recovered from an unsupported command from the host. Used to reset stall before the next USB transfer. Used when the host issues a clear HALT/Feature request to reset stall.
- bool USB_EndpointIsStalled (USB_PIPE_t pipe)
  Helper function to return the endpoint Stall condition.
- RETURN_CODE_t USB_EndpointStalledConditionAck (USB_PIPE_t pipe)
  Acknowledges the stall status condition by clearing the Stall Status bit. Used to clear the Stall Status bit after a stall has been detected.
- RETURN_CODE_t USB_DataToggleSet (USB_PIPE_t pipe)
  Sets the Data Toggle bit on an endpoint which is used to ensure correct data sequence. Only used if hardware data toggling is not available. After a successful transaction, toggle the Data Toggle bit. For SETUP transactions, ensure that the SETUP stage clears the Data Toggle bit, while the data stage and status stage set the Data Toggle bit.
- RETURN_CODE_t USB_DataToggleClear (USB_PIPE_t pipe)
  Clears the Data Toggle bit on an endpoint which is used to ensure correct data sequence. Only used if hardware data toggling is not available. After a successful transaction, toggle the Data Toggle bit. For SETUP transactions, ensure that the SETUP stage clears the Data Toggle bit, while the data stage and status stage set the Data Toggle bit.
- RETURN_CODE_t USB_DataToggle (USB_PIPE_t pipe)
  Toggles the Data Toggle bit on an endpoint which is used to ensure correct data sequence. Only used if hardware data toggling is not available. After a successful transaction, toggle the Data Toggle bit. For SETUP transactions, ensure that the SETUP stage clears the Data Toggle bit, while the data stage and status stage set the Data Toggle bit.
- RETURN_CODE_t ConvertEndpointSizeToMask (uint16_t endpointSize, USB_ENDPOINT_t endpointType, uint8_t *endpointMaskPtr)
  Converts an endpoint size in number of bytes into a register setting. Converts the endpoint size bit mask based on the EP_BUFSIZE setting of the endpoint control register.
- RETURN_CODE_t EndpointBufferSet (USB_PIPE_t pipe, uint8_t *bufAddress)
  Configures the endpoint data buffer to a location in RAM for the next transaction.

4.6.41.2 Detailed Description

USBPERIPHERALENDPOINT Peripheral Endpoint Header File

4.6.42 source/source-files/usb_peripheral/usb_peripheral_read_write.c File Reference

API module for usb_peripheral covering low level USB transaction functions.

```
#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>
#include <string.h>
#include <usb_common_elements.h>
```

```
#include <usb_config.h>
#include <usb_peripheral_avr_du.h>
#include <usb_peripheral_endpoint.h>
#include <usb_peripheral_read_write.h>
#include <usb_protocol_headers.h>
```

4.6.42.1 Functions

- RETURN_CODE_t USB_TransactionStart (USB_PIPE_t pipe)
  Starts sending or receiving data on an endpoint by clearing BUSNACK. Used as a final step while setting up a transaction on the bus.
- RETURN_CODE_t USB_TransactionAbort (USB_PIPE_t pipe)
  Aborts the next transaction on an endpoint by setting BUSNACK. Used to stop exchanging data on an endpoint. The device will start NAKing requests from the host after calling this API.
- RETURN_CODE_t USB_TransactionCompleteAck (USB_PIPE_t pipe)
  Acknowledges the Transaction Complete status condition by clearing the Transaction Complete status bit. Used to clear the Transaction Complete status bit after a transaction has successfully completed.
- bool USB_TransactionIsCompleted (void)
  Helper function to return the endpoint transaction complete condition.
- RETURN_CODE_t USB_TransactionCompletedPipeGet (USB_PIPE_t *pipe)
  Returns the pipe address and direction for the latest completed transaction.
- RETURN_CODE_t USB_PipeReset (USB_PIPE_t pipe)
  Resets the pipe.
- USB_TRANSFER_STATUS_t USB_PipeStatusGet (USB_PIPE_t pipe)
  Gets the current status of pipe.
- bool USB_PipeStatusIsBusy (USB_PIPE_t pipe)
  Checks if the pipe status is busy.
- void USB_PipeDataPtrSet (USB_PIPE_t pipe, uint8_t *dataPtr)
  Configures the pointer for the data transfer in a given pipe.
- uint8_t * USB_PipeDataPtrGet (USB_PIPE_t pipe)
  Gets the current data pointer for a given pipe.
- void USB_PipeDataToTransferSizeSet (USB_PIPE_t pipe, uint16_t dataSize)
  Sets the size of pipe data to transfer.
- uint16_t USB_PipeDataToTransferSizeGet (USB_PIPE_t pipe)
  Gets the size of pipe data to transfer.
- uint16_t USB_PipeDataTransferredSizeGet (USB_PIPE_t pipe)
  Gets the size of the transferred pipe data.
- void USB_PipeDataTransferredSizeSet (USB_PIPE_t pipe, uint16_t dataSize)
  Sets the size of the transferred pipe data.
- void USB_PipeDataTransferredSizeReset (USB_PIPE_t pipe)
  Resets the size of transferred pipe data.
- void USB_PipeTransferZLP_Enable (USB_PIPE_t pipe)
  Enables a ZLP on a transfer. It is enabled by default if the AZLP static config is enabled for the pipe.
- void USB_PipeTransferEndCallbackRegister (USB_PIPE_t pipe, USB_TRANSFER_END_CALLBACK_t callback)

Sets the callback for transfer end.

- void USB_PipeTransferEndCallback (USB_PIPE_t pipe)
  Calls the callback for transfer end.
- RETURN_CODE_t USB_InTransactionRun (USB_PIPE_t pipe)
  Checks the correctness of IN transactions and runs them.
- RETURN_CODE_t USB_OutTransactionRun (USB_PIPE_t pipe)
  Checks the correctness OUT transactions and runs them.
- RETURN_CODE_t USB_PipeTransactionComplete (USB_PIPE_t pipe)
  Handles completed IN and OUT transactions. Processes the completed transaction and either completes the transfer or runs the next transaction. Will call the pipe transferEndCallback at the end of transfer, if configured.

4.6.42.2 Macros

- #define PipeTransferIndexGet(pipe) (((pipe).address * 2) + (pipe).direction)

4.6.42.3 Variables

- STATIC USB_PIPE_TRANSFER_t pipeTransfer [USB_EP_NUM *2]

4.6.42.4 Detailed Description

API module for usb_peripheral covering low level USB transaction functions.

USBPERIPHERALREADWRITE Peripheral Read/Write Source File

Version: USB Device Stack HAL Driver Version 1.0.0

4.6.42.5 Macro Definition Documentation

4.6.42.5.1 PipeTransferIndexGet

#define PipeTransferIndexGet(pipe) (((pipe).address * 2) + (pipe).direction)

4.6.42.6 Variable Documentation

4.6.42.6.1 pipeTransfer

STATIC USB_PIPE_TRANSFER_t pipeTransfer[USB_EP_NUM *2]

4.6.43 source/source-files/usb_peripheral/usb_peripheral_read_write.h File Reference

```
#include <stdbool.h>
#include <stdint.h>
#include "usb_common_elements.h"
#include "usb_protocol_headers.h"
```

4.6.43.1 Functions

- RETURN_CODE_t USB_TransactionStart (USB_PIPE_t pipe)
  Starts sending or receiving data on an endpoint by clearing BUSNACK. Used as a final step while setting up a transaction on the bus.
- RETURN_CODE_t USB_TransactionAbort (USB_PIPE_t pipe)
  Aborts the next transaction on an endpoint by setting BUSNACK. Used to stop exchanging data on an endpoint. The device will start NAKing requests from the host after calling this API.
- RETURN_CODE_t USB_TransactionCompleteAck (USB_PIPE_t pipe)
  Acknowledges the Transaction Complete status condition by clearing the Transaction Complete status bit. Used to clear the Transaction Complete status bit after a transaction has successfully completed.
- bool USB_TransactionIsCompleted (void)
  Helper function to return the endpoint transaction complete condition.

- RETURN_CODE_t USB_TransactionCompletedPipeGet (USB_PIPE_t *pipe)
  Returns the pipe address and direction for the latest completed transaction.
- RETURN_CODE_t USB_PipeReset (USB_PIPE_t pipe)
  Resets the pipe.
- USB_TRANSFER_STATUS_t USB_PipeStatusGet (USB_PIPE_t pipe)
  Gets the current status of pipe.
- bool USB_PipeStatusIsBusy (USB_PIPE_t pipe)
  Checks if the pipe status is busy.
- void USB_PipeDataPtrSet (USB_PIPE_t pipe, uint8_t *dataPtr)
  Configures the pointer for the data transfer in a given pipe.
- uint8_t * USB_PipeDataPtrGet (USB_PIPE_t pipe)
  Gets the current data pointer for a given pipe.
- void USB_PipeDataToTransferSizeSet (USB_PIPE_t pipe, uint16_t dataSize)
  Sets the size of pipe data to transfer.
- uint16_t USB_PipeDataToTransferSizeGet (USB_PIPE_t pipe)
  Gets the size of pipe data to transfer.
- uint16_t USB_PipeDataTransferredSizeGet (USB_PIPE_t pipe)
  Gets the size of the transferred pipe data.
- void USB_PipeDataTransferredSizeSet (USB_PIPE_t pipe, uint16_t dataSize)
  Sets the size of the transferred pipe data.
- void USB_PipeDataTransferredSizeReset (USB_PIPE_t pipe)
  Resets the size of transferred pipe data.
- void USB_PipeTransferZLP_Enable (USB_PIPE_t pipe)
  Enables a ZLP on a transfer. It is enabled by default if the AZLP static config is enabled for the pipe.
- void USB_PipeTransferEndCallbackRegister (USB_PIPE_t pipe, USB_TRANSFER_END_CALLBACK_t callback)
  Sets the callback for transfer end.
- void USB_PipeTransferEndCallback (USB_PIPE_t pipe)
  Calls the callback for transfer end.
- RETURN_CODE_t USB_InTransactionRun (USB_PIPE_t pipe)
  Checks the correctness of IN transactions and runs them.
- RETURN_CODE_t USB_OutTransactionRun (USB_PIPE_t pipe)
  Checks the correctness OUT transactions and runs them.
- RETURN_CODE_t USB_PipeTransactionComplete (USB_PIPE_t pipe)
  Handles completed IN and OUT transactions. Processes the completed transaction and either completes the transfer or runs the next transaction. Will call the pipe transferEndCallback at the end of transfer, if configured.

4.6.43.2 Detailed Description

USBPERIPHERALREADWRITE Peripheral Read/Write Header File

4.6.44 source/source-files/usb_protocol_cdc.h File Reference

USB Communications Device Class (CDC) protocol definitions.

```
#include <stdint.h>
#include <usb_config.h>
```

4.6.44.1 Data structures

- struct USB_CDC_HEADER_FUNCTIONAL_DESCRIPTOR_struct
- struct USB_CDC_ACM_FUNCTIONAL_DESCRIPTOR_struct
- struct USB_CDC_COUNTRY_SELECTION_FUNCTIONAL_DESCRIPTOR_struct
- struct USB_CDC_LINE_CODING_struct

4.6.44.2 Typedefs

- typedef enum USB_CDC_INTERFACE_CLASS_enum USB_CDC_INTERFACE_CLASS_t
- typedef enum USB_CDC_COMM_SUBCLASS_enum USB_CDC_COMM_SUBCLASS_t
- typedef enum USB_CDC_COMM_PROTOCOL_enum USB_CDC_COMM_PROTOCOL_t
- typedef enum USB_CDC_DATA_SUBCLASS_enum USB_CDC_DATA_SUBCLASS_t
- typedef enum USB_CDC_DATA_PROTOCOL_enum USB_CDC_DATA_PROTOCOL_t
- typedef enum USB_CDC_REQUEST_ID_enum USB_CDC_REQUEST_ID_t
- typedef enum USB_CDC_NOTIFICATION_ID_enum USB_CDC_NOTIFICATION_ID_t
- typedef enum USB_CDC_FUNCTIONAL_DESCRIPTOR_enum USB_CDC_FUNCTIONAL_DESCRIPTOR_t
- typedef enum USB_CDC_COMM_FUNCTIONAL_DESCRIPTOR_SUBTYPE_enum USB_CDC_COMM_FUNCTIONAL_DESCRIPTOR_SUBTYPE_t
- typedef enum USB_CDC_DATA_FUNCTIONAL_DESCRIPTOR_SUBTYPE_enum USB_CDC_DATA_FUNCTIONAL_DESCRIPTOR_SUBTYPE_t
- typedef struct USB_CDC_HEADER_FUNCTIONAL_DESCRIPTOR_struct USB_CDC_HEADER_FUNCTIONAL_DESCRIPTOR_t
- typedef struct USB_CDC_ACM_FUNCTIONAL_DESCRIPTOR_struct USB_CDC_ACM_FUNCTIONAL_DESCRIPTOR_t
- typedef enum USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_enum USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_t
- typedef struct USB_CDC_COUNTRY_SELECTION_FUNCTIONAL_DESCRIPTOR_struct USB_CDC_COUNTRY_SELECTION_FUNCTIONAL_DESCRIPTOR_t
- typedef enum USB_CDC_LINE_CODING_STOP_BITS_enum USB_CDC_LINE_CODING_STOP_BITS_t
- typedef enum USD_CDC_LINE_CODING_PARITY_enum USD_CDC_LINE_CODING_PARITY_t
- typedef enum USD_CDC_LINE_CODING_DATA_BITS_enum USD_CDC_LINE_CODING_DATA_BITS_t
- typedef struct USB_CDC_LINE_CODING_struct USB_CDC_LINE_CODING_t
- typedef enum USD_CDC_CONTROL_LINE_STATE_enum USD_CDC_CONTROL_LINE_STATE_t

4.6.44.3 Enumerations

- enum USB_CDC_INTERFACE_CLASS_enum { USB_CDC_NO_INTERFACE_CLASS = 0x00, USB_CDC_COMMUNICATION_INTERFACE_CLASS = 0x02, USB_CDC_DATA_INTERFACE_CLASS = 0x0A }
- enum USB_CDC_COMM_SUBCLASS_enum { USB_CDC_COMM_SUBCLASS_DIRECT_LINE_CONTROL_MODE = 0x01, USB_CDC_COMM_SUBCLASS_ABSTRACT_CONTROL_MODEL = 0x02, USB_CDC_COMM_SUBCLASS_TELEPHONE_CONTROL_MODEL =

0x03, USB_CDC_COMM_SUBCLASS_MULTI_CHANNEL_CONTROL_MODEL = 0x04, USB_CDC_COMM_SUBCLASS_CAPI_CONTROL_MODEL = 0x05, USB_CDC_COMM_SUBCLASS_ETHERNET_NETWORKING_CONTROL_MODEL = 0x06, USB_CDC_COMM_SUBCLASS_ATM_NETWORKING_CONTROL_MODEL = 0x07, USB_CDC_COMM_SUBCLASS_WIRELESS_HANDSET_CONTROL_MODEL = 0x08, USB_CDC_COMM_SUBCLASS_DEVICE_MANAGEMENT = 0x09, USB_CDC_COMM_SUBCLASS_MOBILE_DIRECT_LINE_MODEL = 0x0A, USB_CDC_COMM_SUBCLASS_OBEX = 0x0B, USB_CDC_COMM_SUBCLASS_ETHERNET_EMULATION_MODEL = 0x0C, USB_CDC_COMM_SUBCLASS_NETWORK_CONTROL_MODEL = 0x0D }

- enum USB_CDC_COMM_PROTOCOL_enum { USB_CDC_COMM_NO_PROTOCOL = 0x00, USB_CDC_COMM_PROTOCOL_USB_SPECIFICATION_COMMUNICATIONS = 0x00, USB_CDC_COMM_PROTOCOL_AT_COMMAND_ITU_T_V_250 = 0x01, USB_CDC_COMM_PROTOCOL_AT_COMMAND_PCCA_101 = 0x02, USB_CDC_COMM_PROTOCOL_AT_COMMAND_PCCA_101_ANNEX_O = 0x03, USB_CDC_COMM_PROTOCOL_AT_COMMAND_GSM_7_07 = 0x04, USB_CDC_COMM_PROTOCOL_AT_COMMAND_3GPP_27_07 = 0x05, USB_CDC_COMM_PROTOCOL_AT_COMMAND_C_S0017_0 = 0x06, USB_CDC_COMM_PROTOCOL_USB_EEM = 0x07, USB_CDC_COMM_PROTOCOL_USB_EXTERNAL_PROTOCOL = 0xFE, USB_CDC_COMM_PROTOCOL_USB_SPECIFICATION_VENDOR = 0xFF }
- enum USB_CDC_DATA_SUBCLASS_enum { USB_CDC_DATA_NO_SUBCLASS = 0x00 }
- enum USB_CDC_DATA_PROTOCOL_enum { USB_CDC_DATA_NO_PROTOCOL = 0x00, USB_CDC_DATA_PROTOCOL_USB_SPECIFICATION_DATA = 0x00, USB_CDC_DATA_PROTOCOL_USBNCM1_0 = 0x01, USB_CDC_DATA_PROTOCOL_I_430 = 0x30, USB_CDC_DATA_PROTOCOL_ISO_IEC_3309_1993 = 0x31, USB_CDC_DATA_PROTOCOL_TRANSPARENT = 0x32, USB_CDC_DATA_PROTOCOL_Q_921M = 0x50, USB_CDC_DATA_PROTOCOL_Q_921 = 0x51, USB_CDC_DATA_PROTOCOL_Q_921TM = 0x52, USB_CDC_DATA_PROTOCOL_V_42BIS = 0x90, USB_CDC_DATA_PROTOCOL_Q_931_EURO_ISDN = 0x91, USB_CDC_DATA_PROTOCOL_V_120 = 0x92, USB_CDC_DATA_PROTOCOL_CAPI2_0 = 0x93, USB_CDC_DATA_PROTOCOL_HOST_BASED_DRIVER = 0xFD, USB_CDC_DATA_PROTOCOL_CDC_SPECIFICATION_DATA = 0xFE, USB_CDC_DATA_PROTOCOL_USB_SPECIFICATION_DATA_VENDOR = 0xFF }
- enum USB_CDC_REQUEST_ID_enum { USB_CDC_REQUEST_SEND_ENCAPSULATED_COMMAND = 0x00, USB_CDC_REQUEST_GET_ENCAPSULATED_RESPONSE = 0x01, USB_CDC_REQUEST_SET_COMM_FEATURE = 0x02, USB_CDC_REQUEST_GET_COMM_FEATURE = 0x03, USB_CDC_REQUEST_CLEAR_COMM_FEATURE = 0x04, USB_CDC_REQUEST_SET_AUX_LINE_STATE = 0x10, USB_CDC_REQUEST_SET_HOOK_STATE = 0x11, USB_CDC_REQUEST_PULSE_SETUP = 0x12, USB_CDC_REQUEST_SEND_PULSE = 0x13, USB_CDC_REQUEST_SET_PULSE_TIME = 0x14, USB_CDC_REQUEST_RING_AUX_JACK = 0x15, USB_CDC_REQUEST_SET_LINE_CODING = 0x20, USB_CDC_REQUEST_GET_LINE_CODING = 0x21, USB_CDC_REQUEST_SET_CONTROL_LINE_STATE = 0x22, USB_CDC_REQUEST_SEND_BREAK = 0x23, USB_CDC_REQUEST_SET_RINGER_PARMS = 0x30, USB_CDC_REQUEST_GET_RINGER_PARMS = 0x31, USB_CDC_REQUEST_SET_OPERATION_PARMS = 0x32, USB_CDC_REQUEST_GET_OPERATION_PARMS = 0x33, USB_CDC_REQUEST_SET_LINE_PARMS = 0x34, USB_CDC_REQUEST_GET_LINE_PARMS = 0x35, USB_CDC_REQUEST_DIAL_DIGITS = 0x36, USB_CDC_REQUEST_SET_UNIT_PARAMETER = 0x37, USB_CDC_REQUEST_GET_UNIT_PARAMETER = 0x38, USB_CDC_REQUEST_CLEAR_UNIT_PARAMETER = 0x39, USB_CDC_REQUEST_GET_PROFILE = 0x3A, USB_CDC_REQUEST_SET_ETHERNET_MULTICAST_FILTERS = 0x40, USB_CDC_REQUEST_SET_ETHERNET_POWER_MANAGEMENT_PATTERN_FILTER = 0x41, USB_CDC_REQUEST_GET_ETHERNET_POWER_MANAGEMENT_PATTERN_FILTER = 0x42, USB_CDC_REQUEST_SET_ETHERNET_PACKET_FILTER = 0x43, USB_CDC_REQUEST_GET_ETHERNET_STATISTIC =

0x44, USB_CDC_REQUEST_SET_ATM_DATA_FORMAT = 0x50,
USB_CDC_REQUEST_GET_ATM_DEVICE_STATISTICS = 0x51,
USB_CDC_REQUEST_SET_ATM_DEFAULT_VC = 0x52, USB_CDC_REQUEST_GET_ATM_VC_STATISTICS
= 0x53, USB_CDC_REQUEST_MDLM_SEMANTIC_MODEL_SPECIFIC_REQUESTS
= 0x60, USB_CDC_REQUEST_GET_NTB_PARAMETERS = 0x80,
USB_CDC_REQUEST_GET_NET_ADDRESS = 0x81, USB_CDC_REQUEST_SET_NET_ADDRESS = 0x82,
USB_CDC_REQUEST_GET_NTB_FORMAT = 0x83, USB_CDC_REQUEST_SET_NTB_FORMAT = 0x84,
USB_CDC_REQUEST_GET_NTB_INPUT_SIZE = 0x85, USB_CDC_REQUEST_SET_NTB_INPUT_SIZE
= 0x86, USB_CDC_REQUEST_GET_MAX_DATAGRAM_SIZE = 0x87,
USB_CDC_REQUEST_SET_MAX_DATAGRAM_SIZE = 0x88, USB_CDC_REQUEST_GET_CRC_MODE =
0x89, USB_CDC_REQUEST_SET_CRC_MODE = 0x8A }

- enum USB_CDC_NOTIFICATION_ID_enum { USB_CDC_NOTIFICATION_NETWORK_CONNECTION
= 0x00, USB_CDC_NOTIFICATION_RESPONSE_AVAILABLE =
0x01, USB_CDC_NOTIFICATION_AUX_JACK_HOOK_STATE = 0x08,
USB_CDC_NOTIFICATION_RING_DETECT = 0x09, USB_CDC_NOTIFICATION_SERIAL_STATE
= 0x20, USB_CDC_NOTIFICATION_CALL_STATE_CHANGE =
0x28, USB_CDC_NOTIFICATION_LINE_STATE_CHANGE = 0x29,
USB_CDC_NOTIFICATION_CONNECTION_SPEED_CHANGE = 0x2A,
USB_CDC_NOTIFICATION_MDML_SEMANTIC_MODEL_SPECIFIC_NOTIFICATION = 0x40 }
- enum USB_CDC_FUNCTIONAL_DESCRIPTOR_enum { USB_CDC_FD_CS_INTERFACE = 0x24,
USB_CDC_FD_CS_ENDPOINT = 0x25 }
- enum USB_CDC_COMM_FUNCTIONAL_DESCRIPTOR_SUBTYPE_enum
{ USB_CDC_COMM_FD_SUBTYPE_HEADER = 0x00,
USB_CDC_COMM_FD_SUBTYPE_CALL_MANAGEMENT = 0x01,
USB_CDC_COMM_FD_SUBTYPE_ABSTRACT_CONTROL_MANAGEMENT = 0x02,
USB_CDC_COMM_FD_SUBTYPE_DIRECT_LINE_MANAGEMENT = 0x03,
USB_CDC_COMM_FD_SUBTYPE_TELEPHONE_RINGER = 0x04,
USB_CDC_COMM_FD_SUBTYPE_TELEPHONE_CALL_AND_LINE_STATE_REPORTING_CAPABILITIES
= 0x05, USB_CDC_COMM_FD_SUBTYPE_UNION =
0x06, USB_CDC_COMM_FD_SUBTYPE_COUNTRY_SELECTION =
0x07, USB_CDC_COMM_FD_SUBTYPE_TELEPHONE_OPERATIONAL_MODES
= 0x08, USB_CDC_COMM_FD_SUBTYPE_USB_TERMINAL =
0x09, USB_CDC_COMM_FD_SUBTYPE_NETWORK_CHANNEL =
0x0A, USB_CDC_COMM_FD_SUBTYPE_PROTOCOL_UNIT = 0x0B,
USB_CDC_COMM_FD_SUBTYPE_EXTENSION_UNIT = 0x0C,
USB_CDC_COMM_FD_SUBTYPE_MULTI_CHANNEL_MANAGEMENT = 0x0D,
USB_CDC_COMM_FD_SUBTYPE_CAPI_CONTROL_MANAGEMENT = 0x0E,
USB_CDC_COMM_FD_SUBTYPE_ETHERNET_NETWORKING = 0x0F,
USB_CDC_COMM_FD_SUBTYPE_ATM_NETWORKING = 0x10,
USB_CDC_COMM_FD_SUBTYPE_WIRELESS_HANDSET_CONTROL_MODEL =
0x11, USB_CDC_COMM_FD_SUBTYPE_MOBILE_DIRECT_LINE_MODEL =
0x12, USB_CDC_COMM_FD_SUBTYPE_MDLM_DETAIL = 0x13,
USB_CDC_COMM_FD_SUBTYPE_DEVICE_MANAGEMENT_MODEL = 0x14,
USB_CDC_COMM_FD_SUBTYPE_OBEX = 0x15, USB_CDC_COMM_FD_SUBTYPE_COMMAND_SET
= 0x16, USB_CDC_COMM_FD_SUBTYPE_COMMAND_SET_DETAIL =
0x17, USB_CDC_COMM_FD_SUBTYPE_TELEPHONE_CONTROL_MODEL =
0x18, USB_CDC_COMM_FD_SUBTYPE_OBEX_SERVICE_IDENTIFIER = 0x19,
USB_CDC_COMM_FD_SUBTYPE_NCM = 0x1A }
- enum USB_CDC_DATA_FUNCTIONAL_DESCRIPTOR_SUBTYPE_enum { USB_CDC_DATA_FD_HEADER
= 0x00 }
- enum USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_enum
{ USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_COMM_FEATURE = 0x01,
USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_CONTROL_LINE_CODING = 0x02,

USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_SEND_BREAK_bm = 0x04, USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_NETWORK_CONNECTION_bm = 0x08 }

- enum USB_CDC_LINE_CODING_STOP_BITS_enum { USB_CDC_LINE_CODING_ONE_STOP_BIT = 0x00, USB_CDC_LINE_CODING_ONE_AND_ONE_HALF_STOP_BIT = 0x01, USB_CDC_LINE_CODING_TWO_STOP_BITS = 0x02 }
- enum USD_CDC_LINE_CODING_PARITY_enum { USB_CDC_LINE_CODING_PARITY_NONE = 0x00, USB_CDC_LINE_CODING_PARITY_ODD = 0x01, USB_CDC_LINE_CODING_PARITY_EVEN = 0x02, USB_CDC_LINE_CODING_PARITY_MARK = 0x03, USB_CDC_LINE_CODING_PARITY_SPACE = 0x04 }
- enum USD_CDC_LINE_CODING_DATA_BITS_enum { USB_CDC_LINE_CODING_5_DATA_BITS = 0x05, USB_CDC_LINE_CODING_6_DATA_BITS = 0x06, USB_CDC_LINE_CODING_7_DATA_BITS = 0x07, USB_CDC_LINE_CODING_8_DATA_BITS = 0x08, USB_CDC_LINE_CODING_16_DATA_BITS = 0x10 }
- enum USD_CDC_CONTROL_LINE_STATE_enum { USB_CDC_DATA_TERMINAL_READY_bm = 0x0001, USB_CDC_REQUEST_TO_SEND_bm = 0x0002 }

4.6.44.4 Detailed Description

USB Communications Device Class (CDC) protocol definitions.

USBPROTOCOLCDC CDC Protocol Header File

Version: USB Device Stack Driver Version 1.0.0

4.6.44.5 Typedef Documentation

4.6.44.5.1 USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_t

typedef enum USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_enum USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_t

4.6.44.5.2 USB_CDC_ACM_FUNCTIONAL_DESCRIPTOR_t

typedef struct USB_CDC_ACM_FUNCTIONAL_DESCRIPTOR_struct USB_CDC_ACM_FUNCTIONAL_DESCRIPTOR_t

4.6.44.5.3 USB_CDC_COMM_FUNCTIONAL_DESCRIPTOR_SUBTYPE_t

typedef enum USB_CDC_COMM_FUNCTIONAL_DESCRIPTOR_SUBTYPE_enum USB_CDC_COMM_FUNCTIONAL_DESCRIPTOR_SUBTYPE_t

4.6.44.5.4 USB_CDC_COMM_PROTOCOL_t

typedef enum USB_CDC_COMM_PROTOCOL_enum USB_CDC_COMM_PROTOCOL_t

4.6.44.5.5 USB_CDC_COMM_SUBCLASS_t

typedef enum USB_CDC_COMM_SUBCLASS_enum USB_CDC_COMM_SUBCLASS_t

4.6.44.5.6 USB_CDC_COUNTRY_SELECTION_FUNCTIONAL_DESCRIPTOR_t

typedef struct USB_CDC_COUNTRY_SELECTION_FUNCTIONAL_DESCRIPTOR_struct USB_CDC_COUNTRY_SELECTION_FUNCTIONAL_DESCRIPTOR_t

4.6.44.5.7 USB_CDC_DATA_FUNCTIONAL_DESCRIPTOR_SUBTYPE_t

typedef enum USB_CDC_DATA_FUNCTIONAL_DESCRIPTOR_SUBTYPE_enum USB_CDC_DATA_FUNCTIONAL_DESCRIPTOR_SUBTYPE_t

4.6.44.5.8 USB_CDC_DATA_PROTOCOL_t

typedef enum USB_CDC_DATA_PROTOCOL_enum USB_CDC_DATA_PROTOCOL_t

4.6.44.5.9 USB_CDC_DATA_SUBCLASS_t

typedef enum USB_CDC_DATA_SUBCLASS_enum USB_CDC_DATA_SUBCLASS_t

4.6.44.5.10 USB_CDC_FUNCTIONAL_DESCRIPTOR_t

typedef enum USB_CDC_FUNCTIONAL_DESCRIPTOR_enum USB_CDC_FUNCTIONAL_DESCRIPTOR_t

4.6.44.5.11 USB_CDC_HEADER_FUNCTIONAL_DESCRIPTOR_t

typedef struct USB_CDC_HEADER_FUNCTIONAL_DESCRIPTOR_struct USB_CDC_HEADER_FUNCTIONAL_DESCRIPTOR_t

4.6.44.5.12 USB_CDC_INTERFACE_CLASS_t

typedef enum USB_CDC_INTERFACE_CLASS_enum USB_CDC_INTERFACE_CLASS_t

4.6.44.5.13 USB_CDC_LINE_CODING_STOP_BITS_t

typedef enum USB_CDC_LINE_CODING_STOP_BITS_enum USB_CDC_LINE_CODING_STOP_BITS_t

4.6.44.5.14 USB_CDC_LINE_CODING_t

typedef struct USB_CDC_LINE_CODING_struct USB_CDC_LINE_CODING_t

4.6.44.5.15 USB_CDC_NOTIFICATION_ID_t

typedef enum USB_CDC_NOTIFICATION_ID_enum USB_CDC_NOTIFICATION_ID_t

4.6.44.5.16 USB_CDC_REQUEST_ID_t

typedef enum USB_CDC_REQUEST_ID_enum USB_CDC_REQUEST_ID_t

4.6.44.5.17 USD_CDC_CONTROL_LINE_STATE_t

typedef enum USD_CDC_CONTROL_LINE_STATE_enum USD_CDC_CONTROL_LINE_STATE_t

4.6.44.5.18 USD_CDC_LINE_CODING_DATA_BITS_t

typedef enum USD_CDC_LINE_CODING_DATA_BITS_enum USD_CDC_LINE_CODING_DATA_BITS_t

4.6.44.5.19 USD_CDC_LINE_CODING_PARITY_t

typedef enum USD_CDC_LINE_CODING_PARITY_enum USD_CDC_LINE_CODING_PARITY_t

4.6.44.6 Enumeration Type Documentation

4.6.44.6.1 USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_enum

enum USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_enum

USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_COMM_FEATURE	Device supports the request combination of Set_Comm_Feature, Clear_Comm_Feature, and Get_Comm_Feature
USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_CONTROL_LINE_CODING	Device supports the request combination of Set_Line_Coding, Set_Control_Line_State, Get_Line_Coding, and the notification Serial_State
USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_SEND_BREAK_bm	Device supports the request Send_Break
USB_CDC_ACM_FUNCTIONAL_CAPABILITIES_NETWORK_CONNECTION_bm	Device supports the notification Network_Connection

4.6.44.6.2 USB_CDC_COMM_FUNCTIONAL_DESCRIPTOR_SUBTYPE_enum

enum USB_CDC_COMM_FUNCTIONAL_DESCRIPTOR_SUBTYPE_enum

USB_CDC_COMM_FD_SUBTYPE_HEADER	Functional descriptor subtype: Header, which marks the beginning of the concatenated set of functional descriptors for the interface
USB_CDC_COMM_FD_SUBTYPE_CALL_MANAGEMENT	Functional descriptor subtype: Call Management
USB_CDC_COMM_FD_SUBTYPE_ABSTRACT_CONTROL_MANAGEMENT	Functional descriptor subtype: Abstract Control Management

USB_CDC_COMM_FD_SUBTYPE_DIRECT_LINE_MANAGEMENT	Functional descriptor subtype: Direct Line Management
USB_CDC_COMM_FD_SUBTYPE_TELEPHONE_RINGER	Functional descriptor subtype: Telephone Ringer
USB_CDC_COMM_FD_SUBTYPE_TELEPHONE_CALL_AND_LINE_STATE_REPORTING_CAPABILITIES	Functional descriptor subtype: Telephone Call and Line State Reporting Capabilities
USB_CDC_COMM_FD_SUBTYPE_UNION	Functional descriptor subtype: Union
USB_CDC_COMM_FD_SUBTYPE_COUNTRY_SELECTION	Functional descriptor subtype: Country Selection
USB_CDC_COMM_FD_SUBTYPE_TELEPHONE_OPERATIONAL_MODES	Functional descriptor subtype: Telephone Operational Modes
USB_CDC_COMM_FD_SUBTYPE_USB_TERMINAL	Functional descriptor subtype: USB Terminal
USB_CDC_COMM_FD_SUBTYPE_NETWORK_CHANNEL	Functional descriptor subtype: Network Channel Terminal
USB_CDC_COMM_FD_SUBTYPE_PROTOCOL_UNIT	Functional descriptor subtype: Protocol Unit
USB_CDC_COMM_FD_SUBTYPE_EXTENSION_UNIT	Functional descriptor subtype: Extension Unit
USB_CDC_COMM_FD_SUBTYPE_MULTI_CHANNEL_MANAGEMENT	Functional descriptor subtype: Multi-Channel Management
USB_CDC_COMM_FD_SUBTYPE_CAPI_CONTROL_MANAGEMENT	Functional descriptor subtype: Common application programming interface Control Management
USB_CDC_COMM_FD_SUBTYPE_ETHERNET_NETWORKING	Functional descriptor subtype: Ethernet Networking
USB_CDC_COMM_FD_SUBTYPE_ATM_NETWORKING	Functional descriptor subtype: ATM Networking
USB_CDC_COMM_FD_SUBTYPE_WIRELESS_HANDSET_CONTROL_MODEL	Functional descriptor subtype: Wireless Handset Control Model
USB_CDC_COMM_FD_SUBTYPE_MOBILE_DIRECT_LINE_MODEL	Functional descriptor subtype: Mobile Direct Line Model
USB_CDC_COMM_FD_SUBTYPE_MDLM_DETAIL	Functional descriptor subtype: Mobile Direct Line Model Detail
USB_CDC_COMM_FD_SUBTYPE_DEVICE_MANAGEMENT_MODEL	Functional descriptor subtype: Device Management Model
USB_CDC_COMM_FD_SUBTYPE_OBEX	Functional descriptor subtype: Object exchange
USB_CDC_COMM_FD_SUBTYPE_COMMAND_SET	Functional descriptor subtype: Command Set

USB_CDC_COMM_FD_SUBTYPE_COMMAND_SET_DETAIL	Functional descriptor subtype: Command Set Detail
USB_CDC_COMM_FD_SUBTYPE_TELEPHONE_CONTROL_MODEL	Functional descriptor subtype: Telephone Control Model
USB_CDC_COMM_FD_SUBTYPE_OBEX_SERVICE_IDENTIFIER	Functional descriptor subtype: Object exchange Service Identifier
USB_CDC_COMM_FD_SUBTYPE_NCM	Functional descriptor subtype: Network control model

4.6.44.6.3 USB_CDC_COMM_PROTOCOL_enum

enum USB_CDC_COMM_PROTOCOL_enum

USB_CDC_COMM_NO_PROTOCOL	Communication interface protocol: No protocol
USB_CDC_COMM_PROTOCOL_USB_SPECIFICATION_COMMUNICATIONS	Communication interface protocol: USB specification communication
USB_CDC_COMM_PROTOCOL_AT_COMMAND_ITU_T_V_250	Communication interface protocol: AT Commands V.250 etc
USB_CDC_COMM_PROTOCOL_AT_COMMAND_PCCA_101	Communication interface protocol: AT Commands defined by PCCA-101
USB_CDC_COMM_PROTOCOL_AT_COMMAND_PCCA_101_ANNEX_O	Communication interface protocol: AT Commands defined by PCCA-101 & Annex O
USB_CDC_COMM_PROTOCOL_AT_COMMAND_GSM_7_07	Communication interface protocol: AT Commands defined by GSM 07.07
USB_CDC_COMM_PROTOCOL_AT_COMMAND_3GPP_27_07	Communication interface protocol: AT Commands defined by 3GPP 27.007
USB_CDC_COMM_PROTOCOL_AT_COMMAND_C_S0017_0	Communication interface protocol: AT Commands defined by TIA for CDMA
USB_CDC_COMM_PROTOCOL_USB_EEM	Communication interface protocol: Ethernet Emulation Model
USB_CDC_COMM_PROTOCOL_USB_EXTERNAL_PROTOCOL	Communication interface protocol: External Protocol, commands defined by Command Set functional descriptor
USB_CDC_COMM_PROTOCOL_USB_SPECIFICATION_VENDOR	Communication interface protocol: Vendor-specific

4.6.44.6.4 USB_CDC_COMM_SUBCLASS_enum

enum USB_CDC_COMM_SUBCLASS_enum

USB_CDC_COMM_SUBCLASS_DIRECT_LINE_CONTROL_MODE	Communication interface subclass: Direct line control model
USB_CDC_COMM_SUBCLASS_ABSTRACT_CONTROL_MODEL	Communication interface subclass: Abstract control model
USB_CDC_COMM_SUBCLASS_TELEPHONE_CONTROL_MODEL	Communication interface subclass: Telephone control model
USB_CDC_COMM_SUBCLASS_MULTI_CHANNEL_CONTROL_MODEL	Communication interface subclass: Multi channel control model
USB_CDC_COMM_SUBCLASS_CAPI_CONTROL_MODEL	Communication interface subclass: Common application programming interface control model
USB_CDC_COMM_SUBCLASS_ETHERNET_NETWORKING_CONTROL_MODEL	Communication interface subclass: Ethernet networking control model

USB_CDC_COMM_SUBCLASS_ATM_NETWORKING_CONTROL_MODEL	Communication interface subclass: ATM networking control model
USB_CDC_COMM_SUBCLASS_WIRELESS_HANDSET_CONTROL_MODEL	Communication interface subclass: Wireless handset control model
USB_CDC_COMM_SUBCLASS_DEVICE_MANAGEMENT	Communication interface subclass: Device management
USB_CDC_COMM_SUBCLASS_MOBILE_DIRECT_LINE_MODEL	Communication interface subclass: Mobile direct line model
USB_CDC_COMM_SUBCLASS_OBEX	Communication interface subclass: Object exchange
USB_CDC_COMM_SUBCLASS_ETHERNET_EMULATION_MODEL	Communication interface subclass: Ethernet emulation model
USB_CDC_COMM_SUBCLASS_NETWORK_CONTROL_MODEL	Communication interface subclass: Network control model

4.6.44.6.5 USB_CDC_DATA_FUNCTIONAL_DESCRIPTOR_SUBTYPE_enum

enum USB_CDC_DATA_FUNCTIONAL_DESCRIPTOR_SUBTYPE_enum

USB_CDC_DATA_FD_HEADER	functional descriptor subtype: Header, which marks the beginning of the concatenated set of functional descriptors for the interface

4.6.44.6.6 USB_CDC_DATA_PROTOCOL_enum

enum USB_CDC_DATA_PROTOCOL_enum

USB_CDC_DATA_NO_PROTOCOL	Data interface protocol: No protocol
USB_CDC_DATA_PROTOCOL_USB_SPECIFICATION_DATA	Data interface protocol: USB specification communication
USB_CDC_DATA_PROTOCOL_USBNCM1_0	Data interface protocol: Network Transfer Block
USB_CDC_DATA_PROTOCOL_I_430	Data interface protocol: Physical interface protocol for ISDN BRI
USB_CDC_DATA_PROTOCOL_ISO_IEC_3309_1993	Data interface protocol: High-level data link control
USB_CDC_DATA_PROTOCOL_TRANSPARENT	Data interface protocol: Transparent
USB_CDC_DATA_PROTOCOL_Q_921M	Data interface protocol: Management protocol for Q.921 data link protocol
USB_CDC_DATA_PROTOCOL_Q_921	Data interface protocol: Data link protocol for Q.931
USB_CDC_DATA_PROTOCOL_Q_921TM	Data interface protocol: TEI-multiplexor for Q.921 data link protocol
USB_CDC_DATA_PROTOCOL_V_42BIS	Data interface protocol: Data compression procedures
USB_CDC_DATA_PROTOCOL_Q_931_EURO_ISDN	Data interface protocol: Euro-ISDN protocol control
USB_CDC_DATA_PROTOCOL_V_120	Data interface protocol: V.24 rate adaptation to ISDN
USB_CDC_DATA_PROTOCOL_CAPI2_0	Data interface protocol: Common application programming interface commands
USB_CDC_DATA_PROTOCOL_HOST_BASED_DRIVER	Data interface protocol: Host based driver
USB_CDC_DATA_PROTOCOL_CDC_SPECIFICATION_DATA	Data interface protocol: Protocol Unit functional descriptors on Communications Class Interface
USB_CDC_DATA_PROTOCOL_USB_SPECIFICATION_DATA_VENDOR	Data interface protocol: Vendor-specific

4.6.44.6.7 USB_CDC_DATA_SUBCLASS_enum

enum USB_CDC_DATA_SUBCLASS_enum

USB_CDC_DATA_NO_SUBCLASS	Data interface subclass: No subclass

4.6.44.6.8 USB_CDC_FUNCTIONAL_DESCRIPTOR_enum

enum USB_CDC_FUNCTIONAL_DESCRIPTOR_enum

USB_CDC_FD_CS_INTERFACE	Functional descriptor is connected to interface
USB_CDC_FD_CS_ENDPOINT	Functional descriptor is connected to endpoint

4.6.44.6.9 USB_CDC_INTERFACE_CLASS_enum

enum USB_CDC_INTERFACE_CLASS_enum

USB_CDC_NO_INTERFACE_CLASS	Interface level: No class
USB_CDC_COMMUNICATION_INTERFACE_CLASS	Interface level: Communication class
USB_CDC_DATA_INTERFACE_CLASS	Interface level: Data class

4.6.44.6.10 USB_CDC_LINE_CODING_STOP_BITS_enum

enum USB_CDC_LINE_CODING_STOP_BITS_enum

USB_CDC_LINE_CODING_ONE_STOP_BIT	Number of stop bits: one
USB_CDC_LINE_CODING_ONE_AND_ONE_HALF_STOP_BIT	Number of stop bits: one and a half
USB_CDC_LINE_CODING_TWO_STOP_BITS	Number of stop bits: two

4.6.44.6.11 USB_CDC_NOTIFICATION_ID_enum

enum USB_CDC_NOTIFICATION_ID_enum

USB_CDC_NOTIFICATION_NETWORK_CONNECTION	Notifies the host about network connection status
USB_CDC_NOTIFICATION_RESPONSE_AVAILABLE	Notifies the host that a response is available
USB_CDC_NOTIFICATION_AUX_JACK_HOOK_STATE	Indicates that the loop has changed on the auxiliary phone interface of the USB device
USB_CDC_NOTIFICATION_RING_DETECT	Indicates ring voltage on the POTS line interface of the USB device
USB_CDC_NOTIFICATION_SERIAL_STATE	Sends asynchronous notification of UART status
USB_CDC_NOTIFICATION_CALL_STATE_CHANGE	Identifies that a change has occurred to the state of a call on the line corresponding to the interface or union for the line
USB_CDC_NOTIFICATION_LINE_STATE_CHANGE	Identifies that a change has occurred to the state of the line corresponding to the interface or main interface of a union sending the notification message
USB_CDC_NOTIFICATION_CONNECTION_SPEED_CHANGE	Informs the host-networking driver that a change in either the uplink or the downlink bit rate of the connection has occurred
USB_CDC_NOTIFICATION_MDML_SEMANTIC_MODEL_SPECIFIC_NOTIFICATION	Notification code 40h-5Fh for MDLM Semantic-Model specific notifications

4.6.44.6.12 USB_CDC_REQUEST_ID_enum

enum USB_CDC_REQUEST_ID_enum

USB_CDC_REQUEST_SEND_ENCAPSULATED_COMMAND	Request code to transmit protocol supported command from host to device
USB_CDC_REQUEST_GET_ENCAPSULATED_RESPONSE	Request code to transmit protocol supported command from device to host
USB_CDC_REQUEST_SET_COMM_FEATURE	Request code to update settings for given communication feature

USB_CDC_REQUEST_GET_COMM_FEATURE	Request code to get the current settings for given communication feature
USB_CDC_REQUEST_CLEAR_COMM_FEATURE	Request code to clear custom settings to given communication feature
USB_CDC_REQUEST_SET_AUX_LINE_STATE	Request code to connect or disconnect a secondary jack to POTS circuit or CODEC
USB_CDC_REQUEST_SET_HOOK_STATE	Request code to set the necessary PSTN line relay code for on-hook, off-hook, and caller ID states
USB_CDC_REQUEST_PULSE_SETUP	Request code to prepare for a pulse-dialing cycle
USB_CDC_REQUEST_SEND_PULSE	Request code to generate a specified number of make/break pulse cycles
USB_CDC_REQUEST_SET_PULSE_TIME	Request code to set the timing of the make and break periods for pulse dialing
USB_CDC_REQUEST_RING_AUX_JACK	Request code to generate a ring signal on a secondary phone jack
USB_CDC_REQUEST_SET_LINE_CODING	Request code to specify typical asynchronous line-character formatting properties
USB_CDC_REQUEST_GET_LINE_CODING	Request code to get the current asynchronous line-character formatting properties
USB_CDC_REQUEST_SET_CONTROL_LINE_STATE	Request code to set RS-232/V.24 style control signals
USB_CDC_REQUEST_SEND_BREAK	Request code to send special carrier modulation that generates an RS-232 style break
USB_CDC_REQUEST_SET_RINGER_PARMS	Request code to configure the ringer for the communications device
USB_CDC_REQUEST_GET_RINGER_PARMS	Request code to return the ringer capabilities of the device and the current status of the device's ringer
USB_CDC_REQUEST_SET_OPERATION_PARMS	Request code to set the operational mode for the device
USB_CDC_REQUEST_GET_OPERATION_PARMS	Request code to get the current operational mode for the device
USB_CDC_REQUEST_SET_LINE_PARMS	Request code to change the state of the line for the given interface
USB_CDC_REQUEST_GET_LINE_PARMS	Request code to report the state of the line for the given interface
USB_CDC_REQUEST_DIAL_DIGITS	Request code to dial the DTMF digits over the specified line
USB_CDC_REQUEST_SET_UNIT_PARAMETER	Request code to set the value of a parameter belonging to a Unit identified by Unit Parameter Structure
USB_CDC_REQUEST_GET_UNIT_PARAMETER	Request code to return the current value of a parameter belonging to a Unit pointed out by Unit Parameter Structure
USB_CDC_REQUEST_CLEAR_UNIT_PARAMETER	Request code to restore the default value of a parameter belonging to a Unit identified by Unit Parameter Structure
USB_CDC_REQUEST_GET_PROFILE	Request code to return the profile information as defined by CAPI 2.0

USB_CDC_REQUEST_SET_ETHERNET_MULTICAST_FILTERS	Request code to set the Ethernet device multicast filters as specified in the sequential list of 48 bit Ethernet multicast addresses
USB_CDC_REQUEST_SET_ETHERNET_POWER_MANAGEMENT_PATTERN_FILTER	Request code to set up the specified Ethernet power management pattern filter as described in the data structure
USB_CDC_REQUEST_GET_ETHERNET_POWER_MANAGEMENT_PATTERN_FILTER	Request code to retrieve the status of the specified Ethernet power management pattern filter from the device
USB_CDC_REQUEST_SET_ETHERNET_PACKET_FILTER	Request code to configure device Ethernet packet filter settings
USB_CDC_REQUEST_GET_ETHERNET_STATISTIC	Request code to retrieve a statistic based on the feature selector
USB_CDC_REQUEST_SET_ATM_DATA_FORMAT	Request code to set the data format selected by the host
USB_CDC_REQUEST_GET_ATM_DEVICE_STATISTICS	Request code to retrieve the device statistics based on the feature selector
USB_CDC_REQUEST_SET_ATM_DEFAULT_VC	Request code to pre-select the VPI/VCI value for subsequent GetATMVCStatistics requests
USB_CDC_REQUEST_GET_ATM_VC_STATISTICS	Request code to retrieve the ATM device statistics based on the feature selector for a pre-selected VPI/VCI as stipulated in latest preceding SetATMDefaultVC request
USB_CDC_REQUEST_MDLM_SEMANTIC_MODEL_SPECIFIC_REQUESTS	Request code 0x60 to 0x7F for MDLM Semantic-Model specific requests
USB_CDC_REQUEST_GET_NTB_PARAMETERS	Request code to retrieve the parameters that describe NTBs for each direction
USB_CDC_REQUEST_GET_NET_ADDRESS	Request code to return the function's current EUI-48 station address
USB_CDC_REQUEST_SET_NET_ADDRESS	Request code to set the function's current EUI-48 station address
USB_CDC_REQUEST_GET_NTB_FORMAT	Request code to return the NTB data format currently being used by the function
USB_CDC_REQUEST_SET_NTB_FORMAT	Request code to select the format of NTB to be used for NTBs transmitted from the function to the host
USB_CDC_REQUEST_GET_NTB_INPUT_SIZE	Request code to return NTB input size currently being used by the function
USB_CDC_REQUEST_SET_NTB_INPUT_SIZE	Request code to select the maximum size of NTB that the device is permitted to send to the host
USB_CDC_REQUEST_GET_MAX_DATAGRAM_SIZE	Request code to return the currently effective maximum datagram size that the function has in effect
USB_CDC_REQUEST_SET_MAX_DATAGRAM_SIZE	Request code to select the maximum datagram size that either host or function will send in an NTB
USB_CDC_REQUEST_GET_CRC_MODE	Request code to return the currently selected CRC mode for NTBs formatted by the function
USB_CDC_REQUEST_SET_CRC_MODE	Request code to control whether the function will append CRCs to datagrams when formatting NTBs to be sent to the host

4.6.44.6.13 USD_CDC_CONTROL_LINE_STATE_enum

enum USD_CDC_CONTROL_LINE_STATE_enum

USB_CDC_DATA_TERMINAL_READY_bm	Indicates to DCE if DTE is present or not. This signal corresponds to V.24 signal 108/2 and RS-232 signal DTR.
USB_CDC_REQUEST_TO_SEND_bm	Carrier control for half duplex modems. This signal corresponds to V.24 signal 105 and RS-232 signal RTS.

4.6.44.6.14 USD_CDC_LINE_CODING_DATA_BITS_enum

enum USD_CDC_LINE_CODING_DATA_BITS_enum

USB_CDC_LINE_CODING_5_DATA_BITS	Data transmission size: five bits
USB_CDC_LINE_CODING_6_DATA_BITS	Data transmission size: six bits
USB_CDC_LINE_CODING_7_DATA_BITS	Data transmission size: seven bits
USB_CDC_LINE_CODING_8_DATA_BITS	Data transmission size: eight bits
USB_CDC_LINE_CODING_16_DATA_BITS	Data transmission size: sixteen bits

4.6.44.6.15 USD_CDC_LINE_CODING_PARITY_enum

enum USD_CDC_LINE_CODING_PARITY_enum

USB_CDC_LINE_CODING_PARITY_NONE	Parity mode: none
USB_CDC_LINE_CODING_PARITY_ODD	Parity mode: odd
USB_CDC_LINE_CODING_PARITY_EVEN	Parity mode: even
USB_CDC_LINE_CODING_PARITY_MARK	Parity mode: mark
USB_CDC_LINE_CODING_PARITY_SPACE	Parity mode: space

4.6.45 source/source-files/usb_protocol_hid.h File Reference

USB Human Interface Device (HID) protocol definitions.

```
#include <usb_common_elements.h>
#include <usb_config.h>
#include <usb_protocol_headers.h>
```

4.6.45.1 Data structures

- struct USB_HID_DESCRIPTOR_t
  Type defines for a standard HID descriptor.
- struct USB_HID_REPORT_DESCRIPTOR_t
  Report descriptor for a HID application.
- struct USB_MOUSE_REPORT_DATA_t
  Type defines for a standard mouse input report.
- struct USB_KEYBOARD_REPORT_DATA_t
  Type defines for a standard keyboard input report.

4.6.45.2 Macros

4.6.45.2.1 HID Class Boot type

Macros for standard HID subclass types.

- #define HID_SUB_CLASS_NOBOOT 0x00
- #define HID_SUB_CLASS_BOOT 0x01

4.6.45.3 HID Protocol types

Macros for standard HID protocol types

- #define HID_PROTOCOL_GENERIC 0x00
- #define HID_PROTOCOL_KEYBOARD 0x01
- #define HID_PROTOCOL_MOUSE 0x02

4.6.45.4 USB descriptor codes

Macros used for USB descriptors.

- #define USB_HID_BCD_V1_11 0x0111
- #define USB_HID_NUM_DESC 0x01

4.6.45.5 HID Country codes

Macros for country codes.

- #define USB_HID_NO_COUNTRY_CODE 0
- #define USB_HID_COUNTRY_ARABIC 1
- #define USB_HID_COUNTRY_BELGIAN 2
- #define USB_HID_COUNTRY_CANADIAN_BILINGUAL 3
- #define USB_HID_COUNTRY_CANADIAN_FRENCH 4
- #define USB_HID_COUNTRY_CZECH_REPUBLIC 5
- #define USB_HID_COUNTRY_DANISH 6
- #define USB_HID_COUNTRY_FINNISH 7
- #define USB_HID_COUNTRY_FRENCH 8
- #define USB_HID_COUNTRY_GERMAN 9
- #define USB_HID_COUNTRY_GREEK 10
- #define USB_HID_COUNTRY_HEBREW 11
- #define USB_HID_COUNTRY_HUNGARY 12
- #define USB_HID_COUNTRY_INTERNATIONAL_ISO 13
- #define USB_HID_COUNTRY_ITALIAN 14
- #define USB_HID_COUNTRY_JAPAN_KATAKANA 15
- #define USB_HID_COUNTRY_KOREAN 16
- #define USB_HID_COUNTRY_LATIN_AMERICAN 17
- #define USB_HID_COUNTRY_NETHERLANDS_DUTCH 18
- #define USB_HID_COUNTRY_NORWEGIAN 19
- #define USB_HID_COUNTRY_PERSIAN_FARSI 20
- #define USB_HID_COUNTRY_POLAND 21
- #define USB_HID_COUNTRY_PORTUGUESE 22
- #define USB_HID_COUNTRY_RUSSIA 23
- #define USB_HID_COUNTRY_SLOVAKIA 24
- #define USB_HID_COUNTRY_SPANISH 25
- #define USB_HID_COUNTRY_SWEDISH 26
- #define USB_HID_COUNTRY_SWISS_FRENCH 27
- #define USB_HID_COUNTRY_SWISS_GERMAN 28
- #define USB_HID_COUNTRY_SWITZERLAND 29
- #define USB_HID_COUNTRY_TAIWAN 30

- #define USB_HID_COUNTRY_TURKISH_Q 31
- #define USB_HID_COUNTRY_UK 32
- #define USB_HID_COUNTRY_US 33
- #define USB_HID_COUNTRY_YUGOSLAVIA 34
- #define USB_HID_COUNTRY_TURKISH_F 35

4.6.45.6 Typedefs

- typedef enum USB_REQUEST_ID_HID_enum USB_REQUEST_ID_HID_t
- typedef enum USB_DESCRIPTOR_TYPE_HID_enum USB_DESCRIPTOR_TYPE_HID_t
- typedef enum USB_HID_ITEM_REPORT_TYPE_enum USB_HID_ITEM_REPORT_TYPE_t
- typedef enum USB_HID_REPORT_TYPE_enum USB_HID_REPORT_TYPE_t
- typedef enum USB_HID_PROTOCOL_enum USB_HID_PROTOCOL_t
- typedef void(* USB_HID_REPORT_CALLBACK_t) (uint16_t report)
  Defines a type for registering a callback for the HID report.

4.6.45.7 Enumerations

- enum USB_REQUEST_ID_HID_enum { USB_REQ_HID_GET_REPORT = 0x01, USB_REQ_HID_GET_IDLE = 0x02, USB_REQ_HID_GET_PROTOCOL = 0x03, USB_REQ_HID_SET_REPORT = 0x09, USB_REQ_HID_SET_IDLE = 0x0A, USB_REQ_HID_SET_PROTOCOL = 0x0B }
- enum USB_DESCRIPTOR_TYPE_HID_enum { USB_DT_HID = 0x21, USB_DT_HID_REPORT = 0x22, USB_DT_HID_PHYSICAL = 0x23 }
- enum USB_HID_ITEM_REPORT_TYPE_enum { USB_HID_ITEM_REPORT_TYPE_MAIN = 0, USB_HID_ITEM_REPORT_TYPE_GLOBAL = 1, USB_HID_ITEM_REPORT_TYPE_LOCAL = 2, USB_HID_ITEM_REPORT_TYPE_LONG = 3 }
- enum USB_HID_REPORT_TYPE_enum { USB_HID_REPORT_TYPE_INPUT = 1, USB_HID_REPORT_TYPE_OUTPUT = 2, USB_HID_REPORT_TYPE_FEATURE = 3 }
- enum USB_HID_PROTOCOL_enum { USB_HID_PROTOCOL_BOOT = 0, USB_HID_PROTOCOL_REPORT = 1 }

4.6.45.8 Detailed Description

USB Human Interface Device (HID) protocol definitions.

USBPROTOCOLHID HID Protocols Header File

Version: USB Device Stack HID Driver Version 1.0.0

4.6.45.9 Macro Definition Documentation

4.6.45.9.1 HID_PROTOCOL_GENERIC

#define HID_PROTOCOL_GENERIC 0x00

4.6.45.9.2 HID_PROTOCOL_KEYBOARD

#define HID_PROTOCOL_KEYBOARD 0x01

4.6.45.9.3 HID_PROTOCOL_MOUSE

#define HID_PROTOCOL_MOUSE 0x02

4.6.45.9.4 HID_SUB_CLASS_BOOT

#define HID_SUB_CLASS_BOOT 0x01

4.6.45.9.5 HID_SUB_CLASS_NOBOOT

#define HID_SUB_CLASS_NOBOOT 0x00

4.6.45.9.6 USB_HID_BCD_V1_11
#define USB_HID_BCD_V1_11 0x0111

4.6.45.9.7 USB_HID_COUNTRY_ARABIC
#define USB_HID_COUNTRY_ARABIC 1

4.6.45.9.8 USB_HID_COUNTRY_BELGIAN
#define USB_HID_COUNTRY_BELGIAN 2

4.6.45.9.9 USB_HID_COUNTRY_CANADIAN_BILINGUAL
#define USB_HID_COUNTRY_CANADIAN_BILINGUAL 3

4.6.45.9.10 USB_HID_COUNTRY_CANADIAN_FRENCH
#define USB_HID_COUNTRY_CANADIAN_FRENCH 4

4.6.45.9.11 USB_HID_COUNTRY_CZECH_REPUBLIC
#define USB_HID_COUNTRY_CZECH_REPUBLIC 5

4.6.45.9.12 USB_HID_COUNTRY_DANISH
#define USB_HID_COUNTRY_DANISH 6

4.6.45.9.13 USB_HID_COUNTRY_FINNISH
#define USB_HID_COUNTRY_FINNISH 7

4.6.45.9.14 USB_HID_COUNTRY_FRENCH
#define USB_HID_COUNTRY_FRENCH 8

4.6.45.9.15 USB_HID_COUNTRY_GERMAN
#define USB_HID_COUNTRY_GERMAN 9

4.6.45.9.16 USB_HID_COUNTRY_GREEK
#define USB_HID_COUNTRY_GREEK 10

4.6.45.9.17 USB_HID_COUNTRY_HEBREW
#define USB_HID_COUNTRY_HEBREW 11

4.6.45.9.18 USB_HID_COUNTRY_HUNGARY
#define USB_HID_COUNTRY_HUNGARY 12

4.6.45.9.19 USB_HID_COUNTRY_INTERNATIONAL_ISO
#define USB_HID_COUNTRY_INTERNATIONAL_ISO 13

4.6.45.9.20 USB_HID_COUNTRY_ITALIAN
#define USB_HID_COUNTRY_ITALIAN 14

4.6.45.9.21 USB_HID_COUNTRY_JAPAN_KATAKANA
#define USB_HID_COUNTRY_JAPAN_KATAKANA 15

4.6.45.9.22 USB_HID_COUNTRY_KOREAN
#define USB_HID_COUNTRY_KOREAN 16

4.6.45.9.23 USB_HID_COUNTRY_LATIN_AMERICAN
#define USB_HID_COUNTRY_LATIN_AMERICAN 17

4.6.45.9.24 USB_HID_COUNTRY_NETHERLANDS_DUTCH
#define USB_HID_COUNTRY_NETHERLANDS_DUTCH 18

4.6.45.9.25 USB_HID_COUNTRY_NORWEGIAN
#define USB_HID_COUNTRY_NORWEGIAN 19

4.6.45.9.26 USB_HID_COUNTRY_PERSIAN_FARSI
#define USB_HID_COUNTRY_PERSIAN_FARSI 20

4.6.45.9.27 USB_HID_COUNTRY_POLAND
#define USB_HID_COUNTRY_POLAND 21

4.6.45.9.28 USB_HID_COUNTRY_PORTUGUESE
#define USB_HID_COUNTRY_PORTUGUESE 22

4.6.45.9.29 USB_HID_COUNTRY_RUSSIA
#define USB_HID_COUNTRY_RUSSIA 23

4.6.45.9.30 USB_HID_COUNTRY_SLOVAKIA
#define USB_HID_COUNTRY_SLOVAKIA 24

4.6.45.9.31 USB_HID_COUNTRY_SPANISH
#define USB_HID_COUNTRY_SPANISH 25

4.6.45.9.32 USB_HID_COUNTRY_SWEDISH
#define USB_HID_COUNTRY_SWEDISH 26

4.6.45.9.33 USB_HID_COUNTRY_SWISS_FRENCH
#define USB_HID_COUNTRY_SWISS_FRENCH 27

4.6.45.9.34 USB_HID_COUNTRY_SWISS_GERMAN
#define USB_HID_COUNTRY_SWISS_GERMAN 28

4.6.45.9.35 USB_HID_COUNTRY_SWITZERLAND
#define USB_HID_COUNTRY_SWITZERLAND 29

4.6.45.9.36 USB_HID_COUNTRY_TAIWAN
#define USB_HID_COUNTRY_TAIWAN 30

4.6.45.9.37 USB_HID_COUNTRY_TURKISH_F
#define USB_HID_COUNTRY_TURKISH_F 35

4.6.45.9.38 USB_HID_COUNTRY_TURKISH_Q
#define USB_HID_COUNTRY_TURKISH_Q 31

4.6.45.9.39 USB_HID_COUNTRY_UK
#define USB_HID_COUNTRY_UK 32

4.6.45.9.40 USB_HID_COUNTRY_US
#define USB_HID_COUNTRY_US 33

4.6.45.9.41 USB_HID_COUNTRY_YUGOSLAVIA
#define USB_HID_COUNTRY_YUGOSLAVIA 34

4.6.45.9.42 USB_HID_NO_COUNTRY_CODE
#define USB_HID_NO_COUNTRY_CODE 0

4.6.45.9.43 USB_HID_NUM_DESC
#define USB_HID_NUM_DESC 0x01

4.6.45.10 Typedef Documentation

4.6.45.10.1 USB_DESCRIPTOR_TYPE_HID_t
typedef enum USB_DESCRIPTOR_TYPE_HID_enum USB_DESCRIPTOR_TYPE_HID_t

4.6.45.10.2 USB_HID_ITEM_REPORT_TYPE_t
typedef enum USB_HID_ITEM_REPORT_TYPE_enum USB_HID_ITEM_REPORT_TYPE_t

4.6.45.10.3 USB_HID_PROTOCOL_t
typedef enum USB_HID_PROTOCOL_enum USB_HID_PROTOCOL_t

4.6.45.10.4 USB_HID_REPORT_TYPE_t

typedef enum USB_HID_REPORT_TYPE_enum USB_HID_REPORT_TYPE_t

4.6.45.10.5 USB_REQUEST_ID_HID_t

typedef enum USB_REQUEST_ID_HID_enum USB_REQUEST_ID_HID_t

4.6.45.11 Enumeration Type Documentation

4.6.45.11.1 USB_DESCRIPTOR_TYPE_HID_enum

enum USB_DESCRIPTOR_TYPE_HID_enum

USB_DT_HID	Descriptor type for HID class. Used to describe HID device information such as country code and HID class specification release.
USB_DT_HID_REPORT	Descriptor type for HID report. Used to define the data format and the controls used by the HID device.
USB_DT_HID_PHYSICAL	Descriptor type for HID physical descriptors (optional). Used to describe the physical characteristics of a device.

4.6.45.11.2 USB_HID_ITEM_REPORT_TYPE_enum

enum USB_HID_ITEM_REPORT_TYPE_enum

USB_HID_ITEM_REPORT_TYPE_MAIN	Main items are used to define the data fields in HID reports, such as Input, Output and Feature reports
USB_HID_ITEM_REPORT_TYPE_GLOBAL	Global items apply to all the subsequent items in the report descriptor and define characteristics such as usage page, logical minimum, logical maximum, etc
USB_HID_ITEM_REPORT_TYPE_LOCAL	Local items define characteristics that are specific to a particular item, such as usage or designator index
USB_HID_ITEM_REPORT_TYPE_LONG	Long items are not commonly used but are available for future expansion and allow for extended data fields that are not defined by the standard HID item types

4.6.45.11.3 USB_HID_PROTOCOL_enum

enum USB_HID_PROTOCOL_enum

USB_HID_PROTOCOL_BOOT	Boot protocol is a simplified protocol that supports a limited set of HID devices
USB_HID_PROTOCOL_REPORT	Report protocol provides a more complex and feature-rich means of communication between the HID device and the host

4.6.45.11.4 USB_HID_REPORT_TYPE_enum

enum USB_HID_REPORT_TYPE_enum

USB_HID_REPORT_TYPE_INPUT	Input reports are sent by the HID device to the host to report the state of the input controls
USB_HID_REPORT_TYPE_OUTPUT	Output reports are sent by the host to the HID device to control the state of the output controls
USB_HID_REPORT_TYPE_FEATURE	Feature reports are used to exchange feature data between the host and the HID device that may not be directly tied to input or output, such as configuration data

4.6.45.11.5 USB_REQUEST_ID_HID_enum

enum USB_REQUEST_ID_HID_enum

USB_REQ_HID_GET_REPORT	Request to get the current report from the HID device
USB_REQ_HID_GET_IDLE	Request to get the current idle rate for a particular HID report
USB_REQ_HID_GET_PROTOCOL	Request to get the current protocol in use by the HID device
USB_REQ_HID_SET_REPORT	Request to send a report to the HID device
USB_REQ_HID_SET_IDLE	Request to set the idle rate for a particular HID report

USB_REQ_HID_SET_PROTOCOL	Request to set the protocol to be used by the HID device

4.6.46 source/source-files/usb_vendor.c File Reference

Contains implementation for USB Vendor drivers.

```
#include <usb_vendor.h>
#include <usb_core.h>
#include <usb_protocol_headers.h>
```

4.6.46.1 Functions

- void USB_VendorClassInit (USB_SETUP_EVENT_CALLBACK_t interfaceEnabledCallback, USB_SETUP_PROCESS_CALLBACK_t vendorControlRequest, USB_EVENT_CALLBACK_t interfaceDisabledCallback)

  Sets up interfaces for use with the Vendor class.

4.6.46.2 Detailed Description

Contains implementation for USB Vendor drivers.

USBVENDOR Vendor Source File

Version: USB Device Stack Driver Version 1.0.0

4.6.47 source/source-files/usb_vendor.h File Reference

```
#include "usb_protocol_headers.h"
```

4.6.47.1 Functions

- void USB_VendorClassInit (USB_SETUP_EVENT_CALLBACK_t interfaceEnabledCallback, USB_SETUP_PROCESS_CALLBACK_t vendorControlRequest, USB_EVENT_CALLBACK_t interfaceDisabledCallback)

  Sets up interfaces for use with the Vendor class.

4.6.47.2 Detailed Description

USBVENDOR Vendor Header File

5. Document Revision History

Note: The document revision is independent of the silicon revision.

5.1 Revision History

Doc. Rev.	Date	Comments
A	05/2024	Initial document release

Microchip Information

The Microchip Website

Microchip provides online support via our website at www.microchip.com/. This website is used to make files and information easily available to customers. Some of the content available includes:

- **Product Support** – Data sheets and errata, application notes and sample programs, design resources, user's guides and hardware support documents, latest software releases and archived software
- **General Technical Support** – Frequently Asked Questions (FAQs), technical support requests, online discussion groups, Microchip design partner program member listing
- **Business of Microchip** – Product selector and ordering guides, latest Microchip press releases, listing of seminars and events, listings of Microchip sales offices, distributors and factory representatives

Product Change Notification Service

Microchip's product change notification service helps keep customers current on Microchip products. Subscribers will receive email notification whenever there are changes, updates, revisions or errata related to a specified product family or development tool of interest.

To register, go to www.microchip.com/pcn and follow the registration instructions.

Customer Support

Users of Microchip products can receive assistance through several channels:

- Distributor or Representative
- Local Sales Office
- Embedded Solutions Engineer (ESE)
- Technical Support

Customers should contact their distributor, representative or ESE for support. Local sales offices are also available to help customers. A listing of sales offices and locations is included in this document.

Technical support is available through the website at: www.microchip.com/support

Microchip Devices Code Protection Feature

Note the following details of the code protection feature on Microchip products:

- Microchip products meet the specifications contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is secure when used in the intended manner, within operating specifications, and under normal conditions.
- Microchip values and aggressively protects its intellectual property rights. Attempts to breach the code protection features of Microchip product is strictly prohibited and may violate the Digital Millennium Copyright Act.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of its code. Code protection does not mean that we are guaranteeing the product is "unbreakable". Code protection is constantly evolving. Microchip is committed to continuously improving the code protection features of our products.

Legal Notice

This publication and the information herein may be used only with Microchip products, including to design, test, and integrate Microchip products with your application. Use of this information in any other manner violates these terms. Information regarding device applications is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure

that your application meets with your specifications. Contact your local Microchip sales office for additional support or, obtain additional support at www.microchip.com/en-us/support/design-help/client-support-services.

THIS INFORMATION IS PROVIDED BY MICROCHIP "AS IS". MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE, OR WARRANTIES RELATED TO ITS CONDITION, QUALITY, OR PERFORMANCE.

IN NO EVENT WILL MICROCHIP BE LIABLE FOR ANY INDIRECT, SPECIAL, PUNITIVE, INCIDENTAL, OR CONSEQUENTIAL LOSS, DAMAGE, COST, OR EXPENSE OF ANY KIND WHATSOEVER RELATED TO THE INFORMATION OR ITS USE, HOWEVER CAUSED, EVEN IF MICROCHIP HAS BEEN ADVISED OF THE POSSIBILITY OR THE DAMAGES ARE FORESEEABLE. TO THE FULLEST EXTENT ALLOWED BY LAW, MICROCHIP'S TOTAL LIABILITY ON ALL CLAIMS IN ANY WAY RELATED TO THE INFORMATION OR ITS USE WILL NOT EXCEED THE AMOUNT OF FEES, IF ANY, THAT YOU HAVE PAID DIRECTLY TO MICROCHIP FOR THE INFORMATION.

Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights unless otherwise stated.

Trademarks

The Microchip name and logo, the Microchip logo, Adaptec, AVR, AVR logo, AVR Freaks, BesTime, BitCloud, CryptoMemory, CryptoRF, dsPIC, flexPWR, HELDO, IGLOO, JukeBlox, KeeLoq, Kleer, LANCheck, LinkMD, maXStylus, maXTouch, MediaLB, megaAVR, Microsemi, Microsemi logo, MOST, MOST logo, MPLAB, OptoLyzer, PIC, picoPower, PICSTART, PIC32 logo, PolarFire, Prochip Designer, QTouch, SAM-BA, SenGenuity, SpyNIC, SST, SST Logo, SuperFlash, Symmetricom, SyncServer, Tachyon, TimeSource, tinyAVR, UNI/O, Vectron, and XMEGA are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

AgileSwitch, ClockWorks, The Embedded Control Solutions Company, EtherSynch, Flashtec, Hyper Speed Control, HyperLight Load, Libero, motorBench, mTouch, Powermite 3, Precision Edge, ProASIC, ProASIC Plus, ProASIC Plus logo, Quiet-Wire, SmartFusion, SyncWorld, TimeCesium, TimeHub, TimePictra, TimeProvider, and ZL are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Adjacent Key Suppression, AKS, Analog-for-the-Digital Age, Any Capacitor, AnyIn, AnyOut, Augmented Switching, BlueSky, BodyCom, Clockstudio, CodeGuard, CryptoAuthentication, CryptoAutomotive, CryptoCompanion, CryptoController, dsPICDEM, dsPICDEM.net, Dynamic Average Matching, DAM, ECAN, Espresso T1S, EtherGREEN, EyeOpen, GridTime, IdealBridge, IGaT, In-Circuit Serial Programming, ICSP, INICnet, Intelligent Paralleling, IntelliMOS, Inter-Chip Connectivity, JitterBlocker, Knob-on-Display, MarginLink, maxCrypto, maxView, memBrain, Mindi, MiWi, MPASM, MPF, MPLAB Certified logo, MPLIB, MPLINK, mSiC, MultiTRAK, NetDetach, Omniscient Code Generation, PICDEM, PICDEM.net, PICkit, PICtail, Power MOS IV, Power MOS 7, PowerSmart, PureSilicon, QMatrix, REAL ICE, Ripple Blocker, RTAX, RTG4, SAM-ICE, Serial Quad I/O, simpleMAP, SimpliPHY, SmartBuffer, SmartHLS, SMART-I.S., storClad, SQI, SuperSwitcher, SuperSwitcher II, Switchtec, SynchroPHY, Total Endurance, Trusted Time, TSHARC, Turing, USBCheck, VariSense, VectorBlox, VeriPHY, ViewSpan, WiperLock, XpressConnect, and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

The Adaptec logo, Frequency on Demand, Silicon Storage Technology, and Symmcom are registered trademarks of Microchip Technology Inc. in other countries.

GestIC is a registered trademark of Microchip Technology Germany II GmbH & Co. KG, a subsidiary of Microchip Technology Inc., in other countries.

ISBN: 978-1-6683-4481-1

Quality Management System

For information regarding Microchip's Quality Management Systems, please visit www.microchip.com/quality.

Worldwide Sales and Service

AMERICAS	ASIA/PACIFIC	ASIA/PACIFIC	EUROPE
Corporate Office 2355 West Chandler Blvd. Chandler, AZ 85224-6199 Tel: 480-792-7200 Fax: 480-792-7277 Technical Support: www.microchip.com/support Web Address: www.microchip.com	**Australia - Sydney** Tel: 61-2-9868-6733	**India - Bangalore** Tel: 91-80-3090-4444	**Austria - Wels** Tel: 43-7242-2244-39 Fax: 43-7242-2244-393
Atlanta Duluth, GA Tel: 678-957-9614 Fax: 678-957-1455	**China - Beijing** Tel: 86-10-8569-7000	**India - New Delhi** Tel: 91-11-4160-8631	**Denmark - Copenhagen** Tel: 45-4485-5910 Fax: 45-4485-2829
Austin, TX Tel: 512-257-3370	**China - Chengdu** Tel: 86-28-8665-5511	**India - Pune** Tel: 91-20-4121-0141	**Finland - Espoo** Tel: 358-9-4520-820
Boston Westborough, MA Tel: 774-760-0087 Fax: 774-760-0088	**China - Chongqing** Tel: 86-23-8980-9588	**Japan - Osaka** Tel: 81-6-6152-7160	**France - Paris** Tel: 33-1-69-53-63-20 Fax: 33-1-69-30-90-79
Chicago Itasca, IL Tel: 630-285-0071 Fax: 630-285-0075	**China - Dongguan** Tel: 86-769-8702-9880	**Japan - Tokyo** Tel: 81-3-6880- 3770	**Germany - Garching** Tel: 49-8931-9700
Dallas Addison, TX Tel: 972-818-7423 Fax: 972-818-2924	**China - Guangzhou** Tel: 86-20-8755-8029	**Korea - Daegu** Tel: 82-53-744-4301	**Germany - Haan** Tel: 49-2129-3766400
Detroit Novi, MI Tel: 248-848-4000	**China - Hangzhou** Tel: 86-571-8792-8115	**Korea - Seoul** Tel: 82-2-554-7200	**Germany - Heilbronn** Tel: 49-7131-72400
Houston, TX Tel: 281-894-5983	**China - Hong Kong SAR** Tel: 852-2943-5100	**Malaysia - Kuala Lumpur** Tel: 60-3-7651-7906	**Germany - Karlsruhe** Tel: 49-721-625370
Indianapolis Noblesville, IN Tel: 317-773-8323 Fax: 317-773-5453 Tel: 317-536-2380	**China - Nanjing** Tel: 86-25-8473-2460	**Malaysia - Penang** Tel: 60-4-227-8870	**Germany - Munich** Tel: 49-89-627-144-0 Fax: 49-89-627-144-44
Los Angeles Mission Viejo, CA Tel: 949-462-9523 Fax: 949-462-9608 Tel: 951-273-7800	**China - Qingdao** Tel: 86-532-8502-7355	**Philippines - Manila** Tel: 63-2-634-9065	**Germany - Rosenheim** Tel: 49-8031-354-560
Raleigh, NC Tel: 919-844-7510	**China - Shanghai** Tel: 86-21-3326-8000	**Singapore** Tel: 65-6334-8870	**Israel - Hod Hasharon** Tel: 972-9-775-5100
New York, NY Tel: 631-435-6000	**China - Shenyang** Tel: 86-24-2334-2829	**Taiwan - Hsin Chu** Tel: 886-3-577-8366	**Italy - Milan** Tel: 39-0331-742611 Fax: 39-0331-466781
San Jose, CA Tel: 408-735-9110 Tel: 408-436-4270	**China - Shenzhen** Tel: 86-755-8864-2200	**Taiwan - Kaohsiung** Tel: 886-7-213-7830	**Italy - Padova** Tel: 39-049-7625286
Canada - Toronto Tel: 905-695-1980 Fax: 905-695-2078	**China - Suzhou** Tel: 86-186-6233-1526	**Taiwan - Taipei** Tel: 886-2-2508-8600	**Netherlands - Drunen** Tel: 31-416-690399 Fax: 31-416-690340
	China - Wuhan Tel: 86-27-5980-5300	**Thailand - Bangkok** Tel: 66-2-694-1351	**Norway - Trondheim** Tel: 47-72884388
	China - Xian Tel: 86-29-8833-7252	**Vietnam - Ho Chi Minh** Tel: 84-28-5448-2100	**Poland - Warsaw** Tel: 48-22-3325737
	China - Xiamen Tel: 86-592-2388138		**Romania - Bucharest** Tel: 40-21-407-87-50
	China - Zhuhai Tel: 86-756-3210040		**Spain - Madrid** Tel: 34-91-708-08-90 Fax: 34-91-708-08-91
			Sweden - Gothenberg Tel: 46-31-704-60-40
			Sweden - Stockholm Tel: 46-8-5090-4654
			UK - Wokingham Tel: 44-118-921-5800 Fax: 44-118-921-5820