Contents | Index

Introduction

Welcome to Microchip's MCHPFSUSB USB firmware framework.

Getting Started:

There are a couple of important folders to point out to those new to the MCHPFSUSB firmware framework. The first the documentation folder associated with this library. This folder contains documentation, application notes, getting started guides, etc about USB and this framework. This folder is located in the "<Install Directory>\Microchip\USB\Documentation" folder. This document is

also located in that folder. Please refer to this folder for additional information.

The MCHPFSUSB firmware framework has the following structure:

- <Install Directory>
 - Microchip
 - <Demo 1>
 - <Demo 2> USB Tools

The Demo1 and Demo2 folders are example user application folders giving showing how to use the various function drivers provided in this library. These folders include the files that would need to be created or modified by a user of this framework.

The Microchip folder contains the framework files and documentation. These files in most cases will not require modification by the user. Users, however, will need to include these framework files into their projects. For more information about the various files and API available in the framework, please see the MCHPFSUSB Device Library topic of this document.

For additional information please see www.microchip.com/usb

Introduction

Contents | Index

Contents | Index | Home

Function Drivers

Topics

Name	Description	
Communication Device Class (CDC)	 The Communication Device Class (CDC) is a USB defined class for communication device. These typically include devices like USB to Serial converters, Modems, USB to Ethernet dongles, etc. For more information about CDC class devices, please see the following documents: AN956 - "Migrating Applications to USB from RS-232 UART with Minin Impact on PC Software" - This document is located in the "<install directory="">\Microchip\USB\Documentation" folder</install> www.usb.org/developers/devclass_docs/CDC1.2_WMC1.1.zip 	
<u>Human</u> <u>Interface</u> <u>Device (HID)</u>	The Human Interface Device (HID) class include devices like mice, keyboard joysticks, game controllers, etc. These devices provide input (and in some cases feedback) providing a user interface to the computer. The HID class cl also be used to create custom devices that don't fall into the typical human interface usage model. Examples of both standard devices and custom HID devices are provided in this framework. The HID driver is found in nearly all operating systems and requires no drive installation on most systems. For more information about HID class devices, please see the following sources: • http://www.usb.org/developers/hidpage	
<u>Mass Storage</u> Device (MSD)	Mass Storage Device (MSD) class are devices that appear like drives when plugged into a host. Examples of these types of devices are thumbdrives (memory sticks), external USB hard disks, or external USB CD drives. MSD drivers can be found in nearly all operating systems and requires no dr installation on most systems. This framework includes several examples examples of MSD examples usin different physical storage media. The MSD implementation in this release us the Microchip Memory Disk (MDD) File System as the physical layer. For more information about MSD class devices, please see the following sources: • http://www.usb.org/developers/devclass_docs/usb_msc_overview_1.2 • http://www.usb.org/developers/devclass_docs/usbmassbulk_10.pdf • http://ww1.microchip.com/downloads/en/AppNotes/01189a.pdf more	

Vendor Class

Vendor Class function drivers are drivers are custom drivers who's functiona is not defined by the USB specification. Examples of vendor class drivers include MCHPUSB (Microchip's custom class driver), WinUSB (provided by Microsoft), and LibUSB (an open source driver).

MCHPFSUSB Device Library > <u>Function Drivers</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Communication Device Class (CDC)

The Communication Device Class (CDC) is a USB defined class for communication device. These typically include devices like USB to Serial converters, Modems, USB to Ethernet dongles, etc.

For more information about CDC class devices, please see the following documents:

- AN956 "Migrating Applications to USB from RS-232 UART with Minimal Impact on PC Software" - This document is located in the "<Install Directory>\Microchip\USB\Documentation" folder
- www.usb.org/developers/devclass_docs/CDC1.2_WMC1.1.zip

Topics

Name	Description
Public API Members	This section includes the API members required to access the CDC function driver
Files	This section lists the files required for use with the device stack. These files should be included in any project using the CDC function driver

MCHPFSUSB Device Library > Function Drivers > Communication Device Class (CDC)

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Public API Members

This section includes the API members required to access the CDC function driver

Topics

Name	Description
Functions and Macros	
Definitions, Constants, and Enums	
Variables	
Depricated API Members	

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Functions and Macros

Functions

	Name	Description
≓∳	<u>CDCInitEP</u>	This function initializes the CDC function driver. This function should be called after the SET_CONFIGURATION command.
≡∳	CDCTxService	CDCTxService handles device-to-host transaction(s). This function should be called once per Main Program loop after the device reaches the configured state.
≡∳	getsUSBUSART	getsUSBUSART copies a string of BYTEs received through USB CDC Bulk OUT endpoint to a user's specified location. It is a non-blocking function. It does not wait for data if there is no data available. Instead it returns '0' to notify the caller that there is no data available.
≡∳	putrsUSBUSART	putrsUSBUSART writes a string of data to the USB including the null character. Use this version, 'putrs', to transfer data literals and data located in program memory.
≓∳	putsUSBUSART	putsUSBUSART writes a string of data to the USB including the null character. Use this version, 'puts', to transfer data from a RAM buffer.
ΞŴ	putUSBUSART	putUSBUSART writes an array of data to the USB. Use this version, is capable of transfering 0x00 (what is typically a NULL character in any of the string transfer functions).

Macros

	Name	Description
~	USBUSARTIsTxTrfReady	This macro is used to check if the CDC class is ready to send more data.
~~O	CDCSetLineCoding	This function is used to manually set the data reported back to the host during a get line coding request. (optional)
	CDCSetBaudRate	This macro is used set the baud rate reported back to

¢		the host during a get line coding request. (optional)
~~O	<u>CDCSetCharacterFormat</u>	This macro is used manually set the character format reported back to the host during a get line coding request. (optional)
↔ 0	<u>CDCSetParity</u>	This function is used manually set the parity format reported back to the host during a get line coding request. (optional)
↔0	<u>CDCSetDataSize</u>	This function is used manually set the number of data bits reported back to the host during a get line coding request. (optional)

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Functions and Macros</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Copy Code

CDCInitEP Function

С

```
void CDCInitEP();
```

Description

This function initializes the CDC function driver. This function sets the default line coding (baud rate, bit parity, number of data bits, and format). This function also enables the endpoints and prepares for the first transfer from the host.

This function should be called after the SET_CONFIGURATION command. This is most simply done by calling this function from the <u>USBCBInitEP()</u> function.

Typical Usage:

```
void USBCBInitEP(void)
{
    CDCInitEP();
}
```

Preconditions

None

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Functions and Macros</u> > <u>CDCInitEP Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\textcircled{$\odot$}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

CDCTxService Function

С

```
void CDCTxService();
```

Description

CDCTxService handles device-to-host transaction(s). This function should be called once per Main Program loop after the device reaches the configured state.

Typical Usage:

```
Copy Code
void main(void)
{
    USBDeviceInit();
    while(1)
    {
        USBDeviceTasks();
        if((USBGetDeviceState() < CONFIGURED_STATE) ||</pre>
            (<u>USBIsDeviceSuspended()</u> == TRUE))
        {
            //Either the device is not configured or we are suspende
            // so we don't want to do execute any application code
            continue; //go back to the top of the while loop
        }
        else
        {
            //Keep trying to send data to the PC as required
            CDCTxService();
            //Run application code.
            UserApplication();
        }
    }
}
```

Preconditions

None

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Functions and Macros</u> > <u>CDCTxService Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

getsUSBUSART Function

```
C
BYTE getsUSBUSART(
char * buffer,
BYTE len
);
```

Description

getsUSBUSART copies a string of BYTEs received through USB CDC Bulk OUT endpoint to a user's specified location. It is a non-blocking function. It does not wait for data if there is no data available. Instead it returns '0' to notify the caller that there is no data available.

Typical Usage:

```
Copy Code
BYTE numBytes;
BYTE buffer[64]
numBytes = getsUSBUSART(buffer,sizeof(buffer)); //until the buff
if(numBytes > 0)
{
    //we received numBytes bytes of data and they are copied int
    // the "buffer" variable. We can do something with the dat
    // here.
}
```

Preconditions

Value of input argument 'len' should be smaller than the maximum endpoint size responsible for receiving bulk data from USB host for CDC class. Input argument 'buffer' should point to a buffer area that is bigger or equal to the size specified by 'len'.

Parameters

Parameters	Description
ĺ	

buffer	Pointer to where received BYTEs are to be stored
len	The number of BYTEs expected.

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Functions and Macros</u> > <u>getsUSBUSART Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

putrsUSBUSART Function

```
C
void putrsUSBUSART(
const ROM char * data
);
```

Description

putrsUSBUSART writes a string of data to the USB including the null character. Use this version, 'putrs', to transfer data literals and data located in program memory.

Typical Usage:

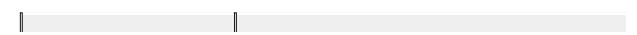
```
Copy Code
if(USBUSARTIsTxTrfReady())
{
    putrsUSBUSART("Hello World");
}
```

The transfer mechanism for device-to-host(put) is more flexible than host-to-device(get). It can handle a string of data larger than the maximum size of bulk IN endpoint. A state machine is used to transfer a long string of data over multiple USB transactions. <u>CDCTxService()</u> must be called periodically to keep sending blocks of data to the host.

Preconditions

USBUSARTIsTxTrfReady() must return TRUE. This indicates that the last transfer is complete and is ready to receive a new block of data. The string of characters pointed to by 'data' must equal to or smaller than 255 BYTEs.

Parameters



Parameters	Description
const ROM char *data	null-terminated string of constant data. If a null character is not found, 255 BYTEs of data will be transferred to the host.

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Functions and Macros</u> > <u>putrsUSBUSART Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

```
USB Device Library Help
```

Contents | Index | Home

Previous | Up | Next

putsUSBUSART Function

```
C
void putsUSBUSART(
char * data
);
```

Description

putsUSBUSART writes a string of data to the USB including the null character. Use this version, 'puts', to transfer data from a RAM buffer.

Typical Usage:

Copy Code

```
if(USBUSARTISTxTrfReady())
{
    char data[] = "Hello World";
    putsUSBUSART(data);
}
```

The transfer mechanism for device-to-host(put) is more flexible than host-to-device(get). It can handle a string of data larger than the maximum size of bulk IN endpoint. A state machine is used to transfer a long string of data over multiple USB transactions. <u>CDCTxService()</u> must be called periodically to keep sending blocks of data to the host.

Preconditions

USBUSARTIsTxTrfReady() must return TRUE. This indicates that the last transfer is complete and is ready to receive a new block of data. The string of characters pointed to by 'data' must equal to or smaller than 255 BYTEs.

Parameters

Parameters

char *data	null-terminated string of constant data. If a null character is not found, 255 BYTEs of data will be transferred to the host.
------------	---

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Functions and Macros</u> > <u>putsUSBUSART Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

```
USB Device Library Help
```

Contents | Index | Home

putUSBUSART Function

```
C
void putUSBUSART(
char * data,
BYTE Length
);
```

Description

putUSBUSART writes an array of data to the USB. Use this version, is capable of transfering 0x00 (what is typically a NULL character in any of the string transfer functions).

Typical Usage:

```
Copy Code
if(USBUSARTIsTxTrfReady())
{
    char data[] = {0x00, 0x01, 0x02, 0x03, 0x04};
    putUSBUSART(data,5);
}
```

The transfer mechanism for device-to-host(put) is more flexible than host-to-device(get). It can handle a string of data larger than the maximum size of bulk IN endpoint. A state machine is used to transfer a long string of data over multiple USB transactions. <u>CDCTxService()</u> must be called periodically to keep sending blocks of data to the host.

Preconditions

<u>USBUSARTIsTxTrfReady</u>() must return TRUE. This indicates that the last transfer is complete and is ready to receive a new block of data. The string of characters pointed to by 'data' must equal to or smaller than 255 BYTEs.

Parameters

Parameters	Description
char *data	pointer to a RAM array of data to be transfered to the host
BYTE length	the number of bytes to be transfered (must be less than 255).

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Functions and Macros</u> > <u>putUSBUSART Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBUSARTIsTxTrfReady Macro

С

#define USBUSARTIsTxTrfReady (cdc_trf_state == CDC_TX_READY)

Description

This macro is used to check if the CDC class is ready to send more data.

Typical Usage:

```
Copy Code
if(USBUSARTIsTxTrfReady())
{
    putrsUSBUSART("Hello World");
}
```

Preconditions

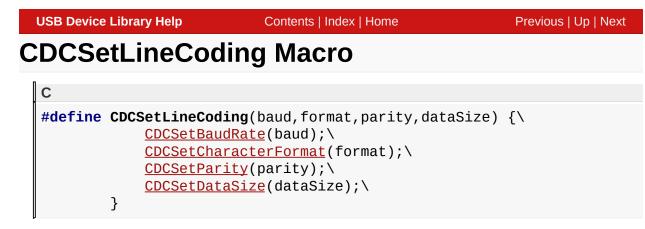
None

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Functions and Macros</u> > <u>USBUSARTISTxTrfReady Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.



Description

This function is used to manually set the data reported back to the host during a get line coding request.

Typical Usage:

```
<u>Copy Code</u>
CDCSetLineCoding(19200, <u>NUM_STOP_BITS_1</u>, <u>PARITY_NONE</u>, 8);
```

This function is optional for CDC devices that do not actually convert the USB traffic to a hardware UART.

Preconditions

None

Parameters

Parameters	Description
DWORD baud	The desired baudrate
BYTE format	 number of stop bits. Available options are: <u>NUM_STOP_BITS_1</u> - 1 Stop bit <u>NUM_STOP_BITS_15</u> - 1.5 Stop bits <u>NUM_STOP_BITS_2</u> - 2 Stop bits

BYTE parity	Type of parity. The options are the following: • <u>PARITY_NONE</u> • <u>PARITY_ODD</u> • <u>PARITY_EVEN</u> • <u>PARITY_MARK</u> • <u>PARITY_SPACE</u>
BYTE dataSize	number of data bits. The options are 5, 6, 7, 8, or 16.

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Functions and Macros</u> > <u>CDCSetLineCoding Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

CDCSetBaudRate Macro

С

#define CDCSetBaudRate(baudRate) {line_coding.dwDTERate.Val=baudRate;}

Description

This macro is used set the baud rate reported back to the host during a get line coding request.

Typical Usage:

Copy Code

```
CDCSetBaudRate(19200);
```

This function is optional for CDC devices that do not actually convert the USB traffic to a hardware UART.

Preconditions

None

Parameters

Parameters	Description
DWORD baudRate	The desired baudrate

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Functions and Macros</u> > <u>CDCSetBaudRate Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008

Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

CDCSetCharacterFormat Macro

С

#define CDCSetCharacterFormat(charFormat) {line_coding.bCharFormat=cha

Description

This macro is used manually set the character format reported back to the host during a get line coding request.

Typical Usage:

Copy Code

CDCSetCharacterFormat(19200);

This function is optional for CDC devices that do not actually convert the USB traffic to a hardware UART.

Preconditions

None

Parameters

Parameters	Description
BYTE charFormat	number of stop bits. Available options are: • <u>NUM_STOP_BITS_1</u> - 1 Stop bit • <u>NUM_STOP_BITS_15</u> - 1.5 Stop bits • <u>NUM_STOP_BITS_2</u> - 2 Stop bits

Remarks

None

MCHPFSUSB Device Library > Function Drivers > Communication Device Class (CDC) > Public

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

CDCSetParity Macro

С

#define CDCSetParity(parityType) {line_coding.bParityType=parityType;}

Description

This macro is used manually set the parity format reported back to the host during a get line coding request.

Typical Usage:

Copy Code

```
CDCSetParity(PARITY_NONE);
```

This function is optional for CDC devices that do not actually convert the USB traffic to a hardware UART.

Preconditions

None

Parameters

Parameters	Description
BYTE parityType	Type of parity. The options are the following: • PARITY_NONE • PARITY_ODD • PARITY_EVEN • PARITY_MARK • PARITY_SPACE

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Functions and Macros</u> > <u>CDCSetParity Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

CDCSetDataSize Macro

С

```
#define CDCSetDataSize(dataBits) {line_coding.bDataBits=dataBits;}
```

Description

This function is used manually set the number of data bits reported back to the host during a get line coding request.

Typical Usage:

Copy Code

```
CDCSetDataSize(8);
```

This function is optional for CDC devices that do not actually convert the USB traffic to a hardware UART.

Preconditions

None

Parameters

Parameters	Description
BYTE dataBits	number of data bits. The options are 5, 6, 7, 8, or 16.

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Functions and Macros</u> > <u>CDCSetDataSize Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008

Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Definitions, Constants, and Enums

Macros

	Name	Description
~~	NUM_STOP_BITS_1	1 stop bit - used by <u>CDCSetLineCoding()</u> and <u>CDCSetCharacterFormat()</u>
~	NUM_STOP_BITS_1_5	1.5 stop bit - used by <u>CDCSetLineCoding()</u> and <u>CDCSetCharacterFormat()</u>
~~	NUM_STOP_BITS_2	2 stop bit - used by <u>CDCSetLineCoding()</u> and <u>CDCSetCharacterFormat()</u>
~~	PARITY_EVEN	even parity - used by <u>CDCSetLineCoding</u> () and <u>CDCSetParity</u> ()
~	PARITY_MARK	mark parity - used by <u>CDCSetLineCoding</u> () and <u>CDCSetParity</u> ()
~	PARITY_NONE	no parity - used by <u>CDCSetLineCoding</u> () and <u>CDCSetParity</u> ()
~	PARITY_ODD	odd parity - used by <u>CDCSetLineCoding()</u> and <u>CDCSetParity()</u>
~	PARITY_SPACE	space parity - used by <u>CDCSetLineCoding()</u> and <u>CDCSetParity()</u>

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Definitions, Constants, and Enums</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

NUM_STOP_BITS_1 Macro

С

#define NUM_STOP_BITS_1 0 //1 stop bit - used by CDCSetLineCoding()

Description

1 stop bit - used by <u>CDCSetLineCoding()</u> and <u>CDCSetCharacterFormat()</u>

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>NUM_STOP_BITS_1 Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

NUM_STOP_BITS_1_5 Macro

С

#define NUM_STOP_BITS_1_5 1 //1.5 stop bit - used by CDCSetLineCodir

Description

1.5 stop bit - used by <u>CDCSetLineCoding()</u> and <u>CDCSetCharacterFormat()</u>

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>NUM_STOP_BITS_1_5 Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

NUM_STOP_BITS_2 Macro

С

#define NUM_STOP_BITS_2 2 //2 stop bit - used by CDCSetLineCoding()

Description

2 stop bit - used by <u>CDCSetLineCoding()</u> and <u>CDCSetCharacterFormat()</u>

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>NUM_STOP_BITS_2 Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

PARITY_EVEN Macro

С

#define PARITY_EVEN 2 //even parity - used by CDCSetLineCoding() and C

Description

even parity - used by <u>CDCSetLineCoding()</u> and <u>CDCSetParity()</u>

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Definitions</u>, <u>Constants</u>, <u>and Enums</u> > <u>PARITY_EVEN Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

PARITY_MARK Macro

С

#define PARITY_MARK 3 //mark parity - used by CDCSetLineCoding() and C

Description

mark parity - used by <u>CDCSetLineCoding()</u> and <u>CDCSetParity()</u>

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Definitions</u>, <u>Constants</u>, <u>and Enums</u> > <u>PARITY_MARK Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

PARITY_NONE Macro

С

#define PARITY_NONE 0 //no parity - used by CDCSetLineCoding() and CDC

Description

no parity - used by <u>CDCSetLineCoding()</u> and <u>CDCSetParity()</u>

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Definitions</u>, <u>Constants</u>, <u>and Enums</u> > <u>PARITY_NONE Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

PARITY_ODD Macro

С

#define PARITY_ODD 1 //odd parity - used by CDCSetLineCoding() and CDC

Description

odd parity - used by <u>CDCSetLineCoding()</u> and <u>CDCSetParity()</u>

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>PARITY_ODD Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

PARITY_SPACE Macro

С

#define PARITY_SPACE 4 //space parity - used by CDCSetLineCoding() and

Description

space parity - used by <u>CDCSetLineCoding()</u> and <u>CDCSetParity()</u>

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Definitions</u>, <u>Constants</u>, <u>and Enums</u> > <u>PARITY_SPACE Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Variables

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Variables</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Depricated API Members

Macros

	Name	Description
~~	mUSBUSARTIsTxTrfReady	Depricated in <u>MCHPFSUSB</u> v2.3. This macro has been replaced by <u>USBUSARTIsTxTrfReady()</u> .

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Depricated API Members</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

mUSBUSARTIsTxTrfReady Macro

С

#define mUSBUSARTIsTxTrfReady USBUSARTIsTxTrfReady()

Description

Depricated in <u>MCHPFSUSB</u> v2.3. This macro has been replaced by <u>USBUSARTIsTxTrfReady()</u>.

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Public</u> <u>API Members</u> > <u>Depricated API Members</u> > <u>mUSBUSARTIsTxTrfReady Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Files

This section lists the files required for use with the device stack. These files should be included in any project using the CDC function driver

Files

Name	Description
usb_function_cdc.h	This file contains all of functions, macros, definitions, variables, datatypes, etc. that are required for usage with the CDC function driver. This file should be included in projects that use the CDC function driver. This file should also be included into the usb_descriptors.c file and any other user file that requires access to the CDC interface. This file is located in the " <install Directory>\Microchip\Include\USB" directory.</install
usb_function_cdc.c	This file contains all of functions, macros, definitions, variables, datatypes, etc. that are required for usage with the CDC function driver. This file should be included in projects that use the CDC function driver. This file is located in the " <install Directory>\Microchip\USB\CDC Device Driver" directory.</install

Topics

Name	Description
usb_config.h	usb_config.h is a file used to configure the <u>MCHPFSUSB</u> stack. This file provides compile time selection of options provided by the stack. This file defines constants needed by the stack and various function drivers.
HardwareProfile.h	HardwareProfile.h is a file used to define hardware specific definitions that are required by the <u>MCHPFSUSB</u> stack. This file should be modified to match the application hardware.

MCHPFSUSB Device Library > Function Drivers > Communication Device Class (CDC) > Files

Copyright © 2008 Microchip Technology, Inc. All rights reserved. $\underline{Contents} \mid \underline{Index} \mid \underline{Home}$

usb_function_cdc.h

USB CDC Function Driver File

This file contains all of functions, macros, definitions, variables, datatypes, etc. that are required for usage with the CDC function driver. This file should be included in projects that use the CDC function driver. This file should also be included into the usb_descriptors.c file and any other user file that requires access to the CDC interface.

This file is located in the "<Install Directory>\Microchip\Include\USB" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\MicrochipInclude

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

Functions

	Name	Description
≡\$	<u>CDCInitEP</u>	This function initializes the CDC function driver. This function should be called after the SET_CONFIGURATION command.
≡\$	CDCTxService	CDCTxService handles device-to-host transaction(s). This function should be called once per Main Program loop after the device reaches the configured state.
≡∳	getsUSBUSART	getsUSBUSART copies a string of BYTEs received through USB CDC Bulk OUT endpoint to a user's specified location. It is a non-blocking function. It does not wait for data if there is no data available. Instead it returns '0' to notify the caller that there is no data available.
≡\$	putrsUSBUSART	putrsUSBUSART writes a string of data to the USB including the null character. Use this version, 'putrs', to transfer data literals and data located in program memory.
≓∳	putsUSBUSART	putsUSBUSART writes a string of data to the USB including the null character. Use this version, 'puts', to transfer data from a RAM buffer.
≓∳	putUSBUSART	putUSBUSART writes an array of data to the USB. Use this version, is capable of transfering 0x00 (what is typically a NULL character in any of the string transfer functions).

Macros

	Name	Description
~~	CDCSetBaudRate	This macro is used set the baud rate reported back to the host during a get line coding request. (optional)
⊷0	<u>CDCSetCharacterFormat</u>	This macro is used manually set the character format reported back to the host during a get line coding request. (optional)
→ O	<u>CDCSetDataSize</u>	This function is used manually set the number of data bits reported back to the host during a get line coding request. (optional)
~	<u>CDCSetLineCoding</u>	This function is used to manually set the data reported back to the host during a get line coding

		request. (optional)
~0	<u>CDCSetParity</u>	This function is used manually set the parity format reported back to the host during a get line coding request. (optional)
~	mUSBUSARTIsTxTrfReady	Depricated in <u>MCHPFSUSB</u> v2.3. This macro has been replaced by <u>USBUSARTIsTxTrfReady()</u> .
~	NUM_STOP_BITS_1	1 stop bit - used by <u>CDCSetLineCoding()</u> and <u>CDCSetCharacterForma</u> t()
~	NUM_STOP_BITS_1_5	1.5 stop bit - used by <u>CDCSetLineCoding</u> () and <u>CDCSetCharacterForma</u> t()
~	NUM_STOP_BITS_2	2 stop bit - used by <u>CDCSetLineCoding()</u> and <u>CDCSetCharacterForma</u> t()
~	PARITY_EVEN	even parity - used by <u>CDCSetLineCoding</u> () and <u>CDCSetParity</u> ()
~	PARITY_MARK	mark parity - used by <u>CDCSetLineCoding()</u> and <u>CDCSetParity()</u>
~	PARITY_NONE	no parity - used by <u>CDCSetLineCoding()</u> and <u>CDCSetParity()</u>
~	PARITY_ODD	odd parity - used by <u>CDCSetLineCoding()</u> and <u>CDCSetParity()</u>
~~	PARITY_SPACE	space parity - used by <u>CDCSetLineCoding()</u> and <u>CDCSetParity()</u>
~	USBUSARTIsTxTrfReady	This macro is used to check if the CDC class is ready to send more data.

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Files</u> > <u>usb_function_cdc.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

usb_function_cdc.c

USB CDC Function Driver File

This file contains all of functions, macros, definitions, variables, datatypes, etc. that are required for usage with the CDC function driver. This file should be included in projects that use the CDC function driver.

This file is located in the "<Install Directory>\Microchip\USB\CDC Device Driver" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\Microchip\Include

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Files</u> > <u>usb_function_cdc.c</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

usb_config.h

usb_config.h is a file used to configure the <u>MCHPFSUSB</u> stack and various function drivers. This file provides compile time selection of options provided by the stack. This file defines constants needed by the stack and various function drivers.

This section will detail the definitions required by the CDC function driver. Additional definitions may be required by the stack itself. Please see the Files topic of the Stack section for more details about stack specific definitions.

Please note that the usb_config.h file can also be generated using the USBConfig.exe tool provided in the "<Install Directory>\USB Tools\USBConfig Tool" directory.

USB_USE_CDC - this definitions indicates to the stack that it will be using the CDC function driver. This needs to be defined for any device using the CDC class.

#define USB_USE_CDC

CDC_COMM_INTF_ID - this defines the interface number of the communication interface used by the CDC function driver. This number should be unique from other interface numbers used in the device for either this class or any other class.

#define CDC_COMM_INTF_ID 0x00

CDC_COMM_EP - this defines the endpoint number of the communication interface used by the CDC function driver. This number should be unique from other endpoint numbers used in the device for either this class or any other class. The valid range for this entry is 1-15.

#define CDC_COMM_EP 2

CDC_COMM_IN_EP_SIZE - this defines the size of the communication endpoint used by the CDC function driver's communication interface.

The current CDC specification only uses 8-byte packets on this endpoint so at this point of time this value should be 8.

CDC_DATA_INTF_ID - this defines the interface number of the data interface used by the CDC function driver. This number should be unique from other interface numbers used in the device for either this class or any other class.

#define CDC_DATA_INTF_ID 0x01

CDC_DATA_EP - this defines the endpoint number of the data interface used by the CDC function driver. This number should be unique from other endpoint numbers used in the device for either this class or any other class. The valid range for this entry is 1-15.

#define CDC_DATA_EP 3

CDC_DATA_OUT_EP_SIZE - this defines the size of the data OUT endpoint used by the CDC function driver's data interface.

#define CDC_DATA_OUT_EP_SIZE 64

CDC_DATA_IN_EP_SIZE - this defines the size of the data IN endpoint used by the CDC function driver's data interface.

#define CDC_DATA_IN_EP_SIZE 64

USB_CDC_SUPPORT_ABSTRACT_CONTROL_MANAGEMENT_CAI - this define tells the CDC function driver to support ACM capabilities D1 (the Set_Line_Coding, Set_Control_Line_State, Get_Line_Coding, and Serial_State commands). For more details please refer to section 5.2.3.3 of the CDC specification (usbcdc11.pdf) available from

www.usb.org. Create this definition if these options are desired.

#define USB_CDC_SUPPORT_ABSTRACT_CONTROL_MANAGEMENT_CAPABILITIES_D1

USB_CDC_SUPPORT_ABSTRACT_CONTROL_MANAGEMENT_CA

- this define tells the CDC function driver to support ACM capabilities D2 (the SEND_BREAK command). For more details please refer to section 5.2.3.3 of the CDC specification (usbcdc11.pdf) available from www.usb.org. Create this definition if these options are desired.

#define USB_CDC_SUPPORT_ABSTRACT_CONTROL_MANAGEMENT_CAPABILITIES_D2

USB_CDC_SUPPORT_HARDWARE_FLOW_CONTROL - this define tells the CDC function driver to implement hardware flow control and UART features. This feature is optional and should only be defined for applications that want this feature. Additional definitions are required in HardwareProfiles.h if this definition is enabled. Please see the HardwareProfiles.h topic in the CDC function driver for more details about those definitions.

#define USB_CDC_SUPPORT_HARDWARE_FLOW_CONTROL

USB_CDC_SET_LINE_CODING_HANDLER - this define tells the CDC function driver which function to call when a set line coding command is received. This function can be used to verify that the line coding is within a supported range and can set the hardware according to the specified settings. If this definition is not defined then no function is called when a set line coding is received and the device will defaulting respond to the host with a successful update of the line coding parameters.

#define USB_CDC_SET_LINE_CODING_HANDLER mySetLineCodingHandler

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Files</u> > <u>usb_config.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

HardwareProfile.h

HardwareProfile.h is a file used to define hardware specific definitions that are required by the <u>MCHPFSUSB</u> stack. This file should be modified to match the application hardware.

The following definitions are required if USB_CDC_SUPPORT_HARDWARE_FLOW_CONTROL is defined in the usb_config.h file. These definitions should be added to the HardwareProfile.h file associated with the project.

UART_TRISTx - this defines the TRIS control bit for the TX pin of the UART module in use.

#define UART_TRISTx TRISBbits.TRISB7

UART_TRISRx - this defines the TRIS control bit for the RX pin of the UART module in use.

#define UART_TRISRx TRISBbits.TRISB5

UART_Tx - this defines the PORT pin control bit for the TX pin of the UART module in use.

#define UART_Tx PORTBbits.RB7

UART_Rx - this defines the PORT pin control bit for the RX pin of the UART module in use.

#define UART_Rx PORTBbits.RB5

UART_TRISRTS - this defines the TRIS control bit for the RTS pin of the UART module in use. Depending on the device selected this may not be part of the module. In this case a general purpose I/O pin should be selected.

#define UART_TRISRTS TRISBbits.TRISB4

UART_RTS - this defines the PORT pin control bit for the RTS pin of the UART module in use. Depending on the device selected this may not be part of the module. In this case a general purpose I/O pin should be selected.

#define UART_RTS PORTBbits.RB4

UART_TRISDTR - this defines the TRIS control bit for the DTR pin of the UART module in use. Depending on the device selected this may not be part of the module. In this case a general purpose I/O pin should be selected.

#define UART_TRISRTS TRISBbits.TRISB4

UART_DTR - this defines the PORT pin control bit for the DTR pin of the UART module in use. Depending on the device selected this may not be part of the module. In this case a general purpose I/O pin should be selected.

#define UART_RTS PORTBbits.RB4

UART_ENABLE - this defines bit in the device that enables the UART module. This bit may vary between devices. Please refer to the appropriate device datasheet for more information.

#define UART_ENABLE RCSTAbits.SPEN

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Communication Device Class (CDC)</u> > <u>Files</u> > <u>HardwareProfile.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Human Interface Device (HID)

The Human Interface Device (HID) class include devices like mice, keyboards, joysticks, game controllers, etc. These devices provide input (and in some cases feedback) providing a user interface to the computer. The HID class can also be used to create custom devices that don't fall into the typical human interface usage model. Examples of both standard devices and custom HID devices are provided in this framework.

The HID driver is found in nearly all operating systems and requires no driver installation on most systems.

For more information about HID class devices, please see the following sources:

http://www.usb.org/developers/hidpage

Topics

Name	Description
Public API Members	This section includes the API members required to access the CDC function driver
Files	This section lists the files required for use with the device stack. These files should be included in any project using the HID function driver

MCHPFSUSB Device Library > Function Drivers > Human Interface Device (HID)

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Public API Members

This section includes the API members required to access the CDC function driver

Topics

Name	Description
Functions and Macros	
Definitions, Constants, and Enums	
Variables	

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Human Interface Device (HID)</u> > <u>Public API</u> <u>Members</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Functions and Macros

Functions

	Name	Description
≓∳	<u>USBCheckHIDRequest</u>	This routine handles HID specific request that happen on EP0. This function should be called from the USBCBCheckOtherReq() call back function whenever implementing a HID device.

Macros

	Name	Description
O	HIDTxPacket	Sends the specified data out the specified endpoint
~~O	HIDTxHandleBusy	Retreives the status of the buffer ownership
~~O	HIDRxPacket	Receives the specified data out the specified endpoint
O	HIDRxHandleBusy	Retreives the status of the buffer ownership

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Human Interface Device (HID)</u> > <u>Public API</u> <u>Members</u> > <u>Functions and Macros</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USBCheckHIDRequest Function

С

void USBCheckHIDRequest();

Description

This routine handles HID specific request that happen on EP0. These include, but are not limited to, requests for the HID report descriptors. This function should be called from the USBCBCheckOtherReg() call back function whenever using an HID device.

Typical Usage:

Copy Code

void USBCBCheckOtherReg(void) //Since the stack didn't handle the request I need to check // my class drivers to see if it is for them USBCheckHIDRequest();

Preconditions

None

{

}

Remarks

None

MCHPFSUSB Device Library > Function Drivers > Human Interface Device (HID) > Public API Members > Functions and Macros > USBCheckHIDRequest Function

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

HIDTxPacket Macro

С

#define HIDTxPacket USBTxOnePacket

Description

This function sends the specified data out the specified endpoint and returns a handle to the transfer information.

Typical Usage:

Copy Code //make sure that the last transfer isn't busy by checking the handle if(!HIDTxHandleBusy(USBInHandle)) { //Send the data contained in the ToSendDataBuffer[] array out on // endpoint HID_EP USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0], siz

Preconditions

None

}

Parameters

Parameters	Description
ер	the endpoint you want to send the data out of
data	pointer to the data that you wish to send
len	the length of the data that you wish to send

Return Values

Return Values	Description

	a handle for the transfer. This information should be kept to track the status of the transfer
--	--

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Human Interface Device (HID)</u> > <u>Public API</u> <u>Members</u> > <u>Functions and Macros</u> > <u>HIDTxPacket Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

HIDTxHandleBusy Macro

С

#define HIDTxHandleBusy(handle) USBHandleBusy(handle)

Description

Retreives the status of the buffer ownership. This function will indicate if the previous transfer is complete or not.

This function will take the input handle (pointer to a BDT entry) and will check the UOWN bit. If the UOWN bit is set then that indicates that the transfer is not complete and the USB module still owns the data memory. If the UOWN bit is clear that means that the transfer is complete and that the CPU now owns the data memory.

For more information about the BDT, please refer to the appropriate datasheet for the device in use.

Typical Usage:

Copy Code
//make sure that the last transfer isn't busy by checking the handle
if(!HIDTxHandleBusy(USBInHandle))
{
 //Send the data contained in the ToSendDataBuffer[] array out on
 // endpoint HID_EP
 USBInHandle = HIDTxPacket(HID_EP,(BYTE*)&ToSendDataBuffer[0],siz
}

Preconditions

None.

Parameters

Parameters	Description
	the handle for the transfer in question. The handle is

	returned by the <u>HIDTxPacket()</u> and <u>HIDRxPacket()</u> functions. Please insure that <u>USB_HANDLE</u> objects are initialized to NULL.

Return Values

Return Values	Description
TRUE	the HID handle is still busy
FALSE	the HID handle is not busy and is ready to send additional data.

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Human Interface Device (HID)</u> > <u>Public API</u> <u>Members</u> > <u>Functions and Macros</u> > <u>HIDTxHandleBusy Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

HIDRxPacket Macro

С

#define HIDRxPacket USBRxOnePacket

Description

Receives the specified data out the specified endpoint.

Typical Usage:

Copy Code

```
//Read 64-bytes from endpoint HID_EP, into the ReceivedDataBuffer ar
// Make sure to save the return handle so that we can check it late
// to determine when the transfer is complete.
USBOutHandle = HIDRxPacket(HID_EP,(BYTE*)&ReceivedDataBuffer,64);
```

Preconditions

None

Parameters

Parameters	Description
ер	the endpoint you want to receive the data into
data	pointer to where the data will go when it arrives
len	the length of the data that you wish to receive

Return Values

Return Values	Description
	a handle for the transfer. This information should be kept to track the status of the transfer

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Human Interface Device (HID)</u> > <u>Public API</u> <u>Members</u> > <u>Functions and Macros</u> > <u>HIDRxPacket Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

HIDRxHandleBusy Macro

С

#define HIDRxHandleBusy(handle) USBHandleBusy(handle)

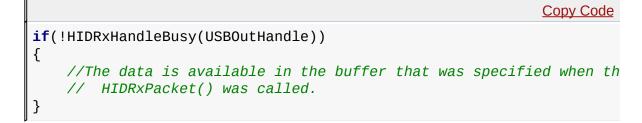
Description

Retreives the status of the buffer ownership. This function will indicate if the previous transfer is complete or not.

This function will take the input handle (pointer to a BDT entry) and will check the UOWN bit. If the UOWN bit is set then that indicates that the transfer is not complete and the USB module still owns the data memory. If the UOWN bit is clear that means that the transfer is complete and that the CPU now owns the data memory.

For more information about the BDT, please refer to the appropriate datasheet for the device in use.

Typical Usage:



Preconditions

None

Return Values

Return Values	Description
TRUE	the HID handle is still busy
	the HID handle is not busy and is ready to receive

FALSE

additional data.

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Human Interface Device (HID)</u> > <u>Public API</u> <u>Members</u> > <u>Functions and Macros</u> > <u>HIDRxHandleBusy Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Definitions, Constants, and Enums

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Human Interface Device (HID)</u> > <u>Public API</u> <u>Members</u> > <u>Definitions, Constants, and Enums</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Variables

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Human Interface Device (HID)</u> > <u>Public API</u> <u>Members</u> > <u>Variables</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Files

This section lists the files required for use with the device stack. These files should be included in any project using the HID function driver

Files

Name	Description
usb_function_hid.c	This file contains all of functions, macros, definitions, variables, datatypes, etc. that are required for usage with the HID function driver. This file should be included in projects that use the HID function driver. This file is located in the " <install Directory>\Microchip\USB\HID Device Driver" directory.</install
usb_function_hid.h	This file contains all of functions, macros, definitions, variables, datatypes, etc. that are required for usage with the HID function driver. This file should be included in projects that use the HID function driver. This file should also be included into the usb_descriptors.c file and any other user file that requires access to the HID interface. This file is located in the " <install Directory>\Microchip\Include\USB" directory.</install

Topics

Name	Description
usb_config.h	usb_config.h is a file used to configure the <u>MCHPFSUSB</u> stack. This file provides compile time selection of options provided by the stack. This file defines constants needed by the stack and various function drivers.

MCHPFSUSB Device Library > Function Drivers > Human Interface Device (HID) > Files

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

usb_function_hid.c

USB HID Function Driver File

This file contains all of functions, macros, definitions, variables, datatypes, etc. that are required for usage with the HID function driver. This file should be included in projects that use the HID function driver.

This file is located in the "<Install Directory>\Microchip\USB\HID Device Driver" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\Microchip\Include

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Human Interface Device (HID)</u> > <u>Files</u> >

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

usb_function_hid.h

USB HID Function Driver File

This file contains all of functions, macros, definitions, variables, datatypes, etc. that are required for usage with the HID function driver. This file should be included in projects that use the HID function driver. This file should also be included into the usb_descriptors.c file and any other user file that requires access to the HID interface.

This file is located in the "<Install Directory>\Microchip\Include\USB" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\MicrochipInclude

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

Functions

	Name	Description
≡∳	USBCheckHIDRequest	This routine handles HID specific request that happen on EP0. This function should be called from the <u>USBCBCheckOtherReg()</u> call back function whenever implementing a HID device.

Macros

	Name	Description
~~	HIDRxHandleBusy	Retreives the status of the buffer ownership
~~	HIDRxPacket	Receives the specified data out the specified endpoint
~~	HIDTxHandleBusy	Retreives the status of the buffer ownership
~~	HIDTxPacket	Sends the specified data out the specified endpoint

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Human Interface Device (HID)</u> > <u>Files</u> > <u>usb_function_hid.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

usb_config.h

usb_config.h is a file used to configure the <u>MCHPFSUSB</u> stack and various function drivers. This file provides compile time selection of options provided by the stack. This file defines constants needed by the stack and various function drivers.

This section will detail the definitions required by the HID function driver. Additional definitions may be required by the stack itself. Please see the Files topic of the Stack section for more details about stack specific definitions.

Please note that the usb_config.h file can also be generated using the USBConfig.exe tool provided in the "<Install Directory>\USB Tools\USBConfig Tool" directory.

USB_USE_HID - This define lets the USB stack know that it will be using the HID function driver. This should be defined for any device using the HID driver.

#define USB_USE_HID

HID_INTF_ID - This defines the interface number for the HID device. This number should be unique within the configuration. Valid numbers range from 0-255.

#define HID_INTF_ID 0x00

HID_EP - This defines the endpoint number used by the HID driver. This number should be unique within the configuration.

#define HID_EP 1

HID_INT_OUT_EP_SIZE - this defines the size of the OUT endpoint used by the HID driver. The valid range is 1-64.

#define HID_INT_OUT_EP_SIZE 3

HID_INT_IN_EP_SIZE - this defines the size of the IN endpoint used by the HID driver. The valid range is 1-64.

#define HID_INT_IN_EP_SIZE 3

HID_NUM_OF_DSC - The HID class specifies its own class descriptors. This defines the number of HID descriptors that this device has in the current configuration.

#define HID_NUM_OF_DSC 1

HID_RPT01_SIZE - This is the size of the first HID report descriptor in bytes.

#define HID_RPT01_SIZE 50

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Human Interface Device (HID)</u> > <u>Files</u> > <u>usb_config.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Mass Storage Device (MSD)

Mass Storage Device (MSD) class are devices that appear like drives when plugged into a host. Examples of these types of devices are thumbdrives (memory sticks), external USB hard disks, or external USB CD drives.

MSD drivers can be found in nearly all operating systems and requires no driver installation on most systems.

This framework includes several examples examples of MSD examples using different physical storage media. The MSD implementation in this release uses the Microchip Memory Disk (MDD) File System as the physical layer.

For more information about MSD class devices, please see the following sources:

- http://www.usb.org/developers/devclass_docs/usb_msc_overview_1.2.pdf
- http://www.usb.org/developers/devclass_docs/usbmassbulk_10.pdf
- http://ww1.microchip.com/downloads/en/AppNotes/01189a.pdf (also located in the documenation folder of this distribution)
- http://ww1.microchip.com/downloads/en/AppNotes/01045b.pdf (also located in the documenation folder of this distribution)

Topics

Name	Description
Public API Members	This section includes the API members required to access the MSD function driver
Files	This section lists the files required for use with the device stack. These files should be included in any project using the MSD function driver In addition to these files additional files may be required from the Microchip MDD File System Library for the physical storage function calls (SD-SPI.c, etc). Please refer to the section discussing the LUN_FUNCTIONS type definition for more details about configuring the library to work with different physical layers in the MDD File System Library.

MCHPFSUSB Device Library > Function Drivers > Mass Storage Device (MSD)

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Public API Members

This section includes the API members required to access the MSD function driver

Topics

Name	Description
Functions and Macros	
Definitions, Constants, and Enums	
Variables	

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Mass Storage Device (MSD)</u> > <u>Public API</u> <u>Members</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Functions and Macros

Functions

	Name	Description
- = \$	<u>MSDTasks</u>	This function runs the MSD class state machines and all of its sub-systems. This function should be called periodically once the device is in the configured state in order to keep the MSD state machine going.
- # \$	<u>USBMSDInit</u>	This routine initializes the MSD class packet handles, prepares to receive a MSD packet, and initializes the MSD state machine. This function should be called once after the device is enumerated.
= 	<u>USBCheckMSDRequest</u>	This routine handles MSD specific request that happen on EP0. This function should be called from the <u>USBCBCheckOtherReq</u> () call back function whenever implementing an MSD device.

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Mass Storage Device (MSD)</u> > <u>Public API</u> <u>Members</u> > <u>Functions and Macros</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

MSDTasks Function

С

```
BYTE MSDTasks();
```

Description

This function runs the MSD class state machines and all of its subsystems. This function should be called periodically once the device is in the configured state in order to keep the MSD state machine going.

Typical Usage:

```
Copy Code
void main(void)
{
    USBDeviceInit();
    while(1)
    {
        USBDeviceTasks();
        if((USBGetDeviceState() < CONFIGURED_STATE) ||</pre>
           (USBIsDeviceSuspended() == TRUE))
        {
            //Either the device is not configured or we are suspende
            // so we don't want to do execute any application code
            continue; //go back to the top of the while loop
        }
        else
        {
            //Keep the MSD state machine going
            MSDTasks();
            //Run application code.
            UserApplication();
        }
    }
}
```

Preconditions

None

Return Values

Return Values	Description
BYTE	the current state of the MSD state machine the valid values are defined in MSD.h under the MSDTasks state machine declaration section. The possible values are the following: • <u>MSD_WAIT</u> • <u>MSD_DATA_IN</u> • <u>MSD_DATA_OUT</u> • <u>MSD_SEND_CSW</u>

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Mass Storage Device (MSD)</u> > <u>Public API</u> <u>Members</u> > <u>Functions and Macros</u> > <u>MSDTasks Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBMSDInit Function

С

```
void USBMSDInit();
```

Description

This routine initializes the MSD class packet handles, prepares to receive a MSD packet, and initializes the MSD state machine. This function should be called once after the device is enumerated.

Typical Usage:

{

}

Copy Code

```
void USBCBInitEP(void)
```

```
USBEnableEndpoint(MSD_DATA_IN_EP,USB_IN_ENABLED|USB_OUT_ENABLED|
USBMSDInit();
```

Preconditions

The device should already be enumerated with a configuration that supports MSD before calling this function.

Paramters: None

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Mass Storage Device (MSD)</u> > <u>Public API</u> <u>Members</u> > <u>Functions and Macros</u> > <u>USBMSDInit Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright @ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USBCheckMSDRequest Function

С

void USBCheckMSDRequest();

Description

This routine handles MSD specific request that happen on EP0. These include, but are not limited to, the standard RESET and GET_MAX_LUN command requests. This function should be called from the <u>USBCBCheckOtherReq</u>() call back function whenever using an MSD device.

Typical Usage:

```
Copy Code
void USBCBCheckOtherReg(void)
{
    //Since the stack didn't handle the request I need to check
    // my class drivers to see if it is for them
    USBCheckMSDRequest();
}
```

Preconditions

None

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Mass Storage Device (MSD)</u> > <u>Public API</u> <u>Members</u> > <u>Functions and Macros</u> > <u>USBCheckMSDRequest Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright @ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Definitions, Constants, and Enums

Macros

	Name	Description
~~	MSD_WAIT	MSD_WAIT is when the MSD state machine is idle (returned by <u>MSDTasks()</u>)
~~	MSD_DATA_IN	MSD_DATA_IN is when the device is sending data (returned by <u>MSDTasks(</u>))
~~	MSD_DATA_OUT	MSD_DATA_OUT is when the device is receiving data (returned by <u>MSDTasks()</u>)
~~	MSD_SEND_CSW	MSD_SEND_CSW is when the device is waiting to send the CSW (returned by <u>MSDTasks()</u>)

Structures

	Name	Description
♦	LUN_FUNCTIONS	LUN_FUNCTIONS is a structure of function pointers that tells the stack where to find each of the physical layer functions it is looking for. This structure needs to be defined for any project for PIC24F or PIC32.

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Mass Storage Device (MSD)</u> > <u>Public API</u> <u>Members</u> > <u>Definitions, Constants, and Enums</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

LUN_FUNCTIONS Structure

с
<pre>typedef struct {</pre>
BYTE (* MediaInitialize)();
DWORD (* ReadCapacity)();
WORD (* ReadSectorSize)();
BYTE (* MediaDetect)();
BYTE (* SectorRead)(DWORD sector_addr, BYTE* buffer);
BYTE (* WriteProtectState)();
BYTE (* SectorWrite)(DWORD sector_addr, BYTE* buffer, BYTE allowWrit
} LUN_FUNCTIONS;

Description

LUN_FUNCTIONS is a structure of function pointers that tells the stack where to find each of the physical layer functions it is looking for. This structure needs to be defined for any project for PIC24F or PIC32.

Typical Usage:

In the above code we are passing the address of the SDSPI functions to the corresponding member of the LUN_FUNCTIONS structure. In the above case we have created an array of LUN_FUNCTIONS structures so that it is possible to have multiple physical layers by merely increasing the MAX_LUN variable and by adding one more set of

entries in the array. Please take caution to insure that each function is in the the correct location in the structure. Incorrect alignment will cause the USB stack to call the incorrect function for a given command.

See the MDD File System Library for additional information about the available physical media, their requirements, and how to use their associated functions.

Members

Members	Description
BYTE (* MediaInitialize)();	Function pointer to the MediaInitialize() function of the physical media being used.
DWORD (* ReadCapacity)();	Function pointer to the ReadCapacity() function of the physical media being used.
WORD (* ReadSectorSize)();	Function pointer to the ReadSectorSize() function of the physical media being used.
BYTE (* MediaDetect)();	Function pointer to the MediaDetect() function of the physical media being used.
BYTE (* SectorRead)(DWORD sector_addr, BYTE* buffer);	Function pointer to the SectorRead() function of the physical media being used.
BYTE (* WriteProtectState)();	Function pointer to the WriteProtectState() function of the physical media being used.
BYTE (* SectorWrite)(DWORD sector_addr, BYTE* buffer, BYTE allowWriteToZero);	Function pointer to the SectorWrite() function of the physical media being used.

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Mass Storage Device (MSD)</u> > <u>Public API</u> <u>Members</u> > <u>Definitions</u>, <u>Constants</u>, <u>and Enums</u> > <u>LUN_FUNCTIONS Structure</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

MSD_WAIT Macro

С

#define MSD_WAIT 0×00

Description

MSD_WAIT is when the MSD state machine is idle (returned by <u>MSDTasks()</u>)

MCHPFSUSB Device Library > Function Drivers > Mass Storage Device (MSD) > Public API Members > Definitions, Constants, and Enums > MSD_WAIT Macro

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

MSD_DATA_IN Macro

С

#define MSD_DATA_IN 0x01

Description

MSD_DATA_IN is when the device is sending data (returned by <u>MSDTasks()</u>)

MCHPFSUSB Device Library > Function Drivers > Mass Storage Device (MSD) > Public API Members > Definitions, Constants, and Enums > MSD_DATA_IN Macro

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

MSD_DATA_OUT Macro

С

#define MSD_DATA_OUT 0x02

Description

MSD_DATA_OUT is when the device is receiving data (returned by MSDTasks())

MCHPFSUSB Device Library > Function Drivers > Mass Storage Device (MSD) > Public API Members > Definitions, Constants, and Enums > MSD_DATA_OUT Macro

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

MSD_SEND_CSW Macro

С

#define MSD_SEND_CSW 0x03

Description

MSD_SEND_CSW is when the device is waiting to send the CSW (returned by <u>MSDTasks()</u>)

MCHPFSUSB Device Library > Function Drivers > Mass Storage Device (MSD) > Public API Members > Definitions, Constants, and Enums > MSD_SEND_CSW Macro

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Variables

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Mass Storage Device (MSD)</u> > <u>Public API</u> <u>Members</u> > <u>Variables</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Files

This section lists the files required for use with the device stack. These files should be included in any project using the MSD function driver

In addition to these files additional files may be required from the Microchip MDD File System Library for the physical storage function calls (SD-SPI.c, etc). Please refer to the section discussing the LUN_FUNCTIONS type definition for more details about configuring the library to work with different physical layers in the MDD File System Library.

Files

Name	Description
usb_function_msd.c	This file contains functions, macros, definitions, variables, datatypes, etc. that are required for use of the MSD function driver. This file should be included in projects that use the MSD function driver. This file is located in the " <install Directory>\Microchip\USB\MSD Device Driver" directory.</install
usb_function_msd.h	This file contains functions, macros, definitions, variables, datatypes, etc. that are required for use of the MSD function driver. This file should be included in projects that use the MSD function driver. This file is located in the " <install Directory>\Microchip\USB\MSD Device Driver" directory.</install

Topics

Name	Description
usb_config.h	usb_config.h is a file used to configure the <u>MCHPFSUSB</u> stack and various function drivers. This file provides compile time selection of options provided by the stack. This file defines constants needed by the stack and various function drivers.

l

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Mass Storage Device (MSD)</u> > <u>Files</u>

IJ

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

usb_function_msd.c

USB MSD Function Driver File

This file contains functions, macros, definitions, variables, datatypes, etc. that are required for use of the MSD function driver. This file should be included in projects that use the MSD function driver.

This file is located in the "<Install Directory>\Microchip\USB\MSD Device Driver" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\Microchip\Include

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

MCHPFSUSB Device Library > Function Drivers > Mass Storage Device (MSD) > Files >

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

usb_function_msd.h

USB MSD Function Driver File

This file contains functions, macros, definitions, variables, datatypes, etc. that are required for use of the MSD function driver. This file should be included in projects that use the MSD function driver.

This file is located in the "<Install Directory>\Microchip\USB\MSD Device Driver" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\Microchip\Include

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

Functions

	Name	Description
≡ ∳	<u>MSDTasks</u>	This function runs the MSD class state machines and all of its sub-systems. This function should be called periodically once the device is in the configured state in order to keep the MSD state machine going.
≓∳	<u>USBCheckMSDRequest</u>	This routine handles MSD specific request that happen on EP0. This function should be called from the <u>USBCBCheckOtherReq</u> () call back function whenever implementing an MSD device.
.≡∳	<u>USBMSDInit</u>	This routine initializes the MSD class packet handles, prepares to receive a MSD packet, and initializes the MSD state machine. This function should be called once after the device is enumerated.

Macros

	Name	Description
~~	MSD_DATA_IN	MSD_DATA_IN is when the device is sending data (returned by <u>MSDTasks(</u>))
~	MSD_DATA_OUT	MSD_DATA_OUT is when the device is receiving data (returned by <u>MSDTasks(</u>))
~	MSD_SEND_CSW	MSD_SEND_CSW is when the device is waiting to send the CSW (returned by <u>MSDTasks()</u>)
~	MSD_WAIT	MSD_WAIT is when the MSD state machine is idle (returned by <u>MSDTasks()</u>)

Structures

	Name	Description
٠	LUN_FUNCTIONS	LUN_FUNCTIONS is a structure of function pointers that tells the stack where to find each of the physical layer functions it is looking for. This structure needs to be defined for any project for PIC24F or PIC32.

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Mass Storage Device (MSD)</u> > <u>Files</u> > <u>usb_function_msd.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

usb_config.h

usb_config.h is a file used to configure the <u>MCHPFSUSB</u> stack and various function drivers. This file provides compile time selection of options provided by the stack. This file defines constants needed by the stack and various function drivers.

This section will detail the definitions required by the MSD function driver. Additional definitions may be required by the stack itself. Please see the Files topic of the Stack section for more details about stack specific definitions.

Please note that the usb_config.h file can also be generated using the USBConfig.exe tool provided in the "<Install Directory>\USB Tools\USBConfig Tool" directory.

USB_USE_MSD - This define lets the USB stack know that it will be using the MSD function driver. This should be defined for any device using the MSD driver.

#define USB_USE_MSD

MSD_INTF_ID - This defines the interface number for the MSD device. This number should be unique within the configuration. Valid numbers range from 0-255.

#define MSD_INTF_ID 0x00

MSD_IN_EP_SIZE - this defines the size of the IN endpoint used by the MSD driver. The valid range is 1-64.

#define MSD_IN_EP_SIZE 64

MSD_OUT_EP_SIZE - this defines the size of the OUT endpoint used by the MSD driver. The valid range is 1-64.

#define MSD_OUT_EP_SIZE 64

MAX_LUN - This defines the largest logical unit number (LUN) of the device (0 indexed). Each LUN will appear to the computer as its own

drive. For one drive, use 0. For sixteen drives, use 15. The valid range is 0-15. For each valid LUN defined here there should be a corresponding entry in the array of <u>LUN_FUNCTIONS</u> variables.

#define MAX_LUN 0

MSD_DATA_IN_EP - This defined the IN endpoint number used by the MSD driver. This number should be unique within the configuration (but can be the same as the MSD_DATA_OUT_EP).

#define MSD_DATA_IN_EP 1

MSD_DATA_OUT_EP - This defined the OUT endpoint number used by the MSD driver. This number should be unique within the configuration (but can be the same as the MSD_DATA_IN_EP).

#define MSD_DATA_OUT_EP 1

MSD_BUFFER_ADDRESS - This defines the address where the MSD RAM buffer will reside. This implementation requires a sector size buffer (typically 512 bytes).

#define MSD_BUFFER_ADDRESS 0x600

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Mass Storage Device (MSD)</u> > <u>Files</u> > <u>usb_config.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Vendor Class

Vendor Class function drivers are drivers are custom drivers who's functionality is not defined by the USB specification. Examples of vendor class drivers include MCHPUSB (Microchip's custom class driver), WinUSB (provided by Microsoft), and LibUSB (an open source driver).

Topics

Name	Description
Public API Members	
Files	This section lists the files required for use with the device stack. These files should be included in any project using vendor function drivers

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Vendor Class</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Public API Members

Topics

Name	Description
Functions and Macros	
Definitions, Constants, and Enums	
Variables	

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Vendor Class</u> > <u>Public API Members</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Functions and Macros

Macros

	Name	Description
~~O	USBGenRead	Receives the specified data out the specified endpoint
~0	<u>USBGenWrite</u>	Sends the specified data out the specified endpoint

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Vendor Class</u> > <u>Public API Members</u> > <u>Functions and Macros</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBGenRead Macro

С

#define USBGenRead(ep,data,len) USBRxOnePacket(ep,data,len)

Description

Receives the specified data out the specified endpoint.

Typical Usage:

Copy Code
//Read 64-bytes from endpoint USBGEN_EP_NUM, into the OUTPacket arra
// Make sure to save the return handle so that we can check it late
// to determine when the transfer is complete.
if(!USBHandleBusy(USBOutHandle))
{
 USBOutHandle = USBGenRead(USBGEN_EP_NUM,(BYTE*)&OUTPacket,64);
}

Preconditions

None

Parameters

Parameters	Description
ер	the endpoint you want to receive the data into
data	pointer to where the data will go when it arrives
len	the length of the data that you wish to receive

Return Values

Return Values	Description
	a handle for the transfer. This information should be kept to track the status of the transfer

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Vendor Class</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBGenRead Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBGenWrite Macro

С

#define USBGenWrite(ep,data,len) USBTxOnePacket(ep,data,len)

Description

This function sends the specified data out the specified endpoint and returns a handle to the transfer information.

Typical Usage:

Copy Code
//make sure that the last transfer isn't busy by checking the handle
if(!USBHandleBusy(USBGenericInHandle))
{
 //Send the data contained in the INPacket[] array out on
 // endpoint USBGEN_EP_NUM
 USBGenericInHandle = USBGenWrite(USBGEN_EP_NUM,(BYTE*)&INPacket[
}

Preconditions

None

Parameters

Parameters	Description
ер	the endpoint you want to send the data out of
data	pointer to the data that you wish to send
len	the length of the data that you wish to send

Return Values

Return Values	Description

a handle for the transfer. This information should be kept to track the status of the transfer

Remarks

None

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Vendor Class</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBGenWrite Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Definitions, Constants, and Enums

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Vendor Class</u> > <u>Public API Members</u> > <u>Definitions, Constants, and Enums</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Variables

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Vendor Class</u> > <u>Public API Members</u> > <u>Variables</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Files

This section lists the files required for use with the device stack. These files should be included in any project using vendor function drivers

Files

Name	Description	
usb_function_generic.c	This file contains functions, macros, definitions, variables, datatypes, etc. that are required for use of vendor class function drivers. This file should be included in projects that use vendor class function drivers. Vendor class function drivers include MCHPUSB (Microchip's custom class driver), WinUSB, and LibUSB. This file is located in the " <install Directory>\Microchip\USB\Generic Device Driver" directory.</install 	
usb_function_generic.h	This file contains all of functions, macros, definitions, variables, datatypes, etc. that are required for usage with vendor class function drivers. This file should be included in projects that use vendor class function drivers. This file should also be included into the usb_descriptors.c file and any other user file that requires access to vendor class interfaces. This file is located in the " <install Directory>\Microchip\Include\USB" directory.</install 	

Topics

Name	Description
<u>usb_config.h</u>	usb_config.h is a file used to configure the <u>MCHPFSUSB</u> stack and various function drivers. This file provides compile time selection of options provided by the stack. This file defines constants needed by the stack and various function drivers.

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Vendor Class</u> > <u>Files</u>

Copyright © 2008 Microchip Technology, Inc. All rights reserved. $\underline{Contents} \mid \underline{Index} \mid \underline{Home}$

usb_function_generic.c

USB Vender Class Custom Driver Header File

This file contains functions, macros, definitions, variables, datatypes, etc. that are required for use of vendor class function drivers. This file should be included in projects that use vendor class function drivers.

This file is located in the "<Install Directory>\Microchip\USB\Generic Device Driver" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\Microchip\Include

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Vendor Class</u> > <u>Files</u> > <u>usb_function_generic.c</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

usb_function_generic.h

USB Vender Class Custom Driver File

This file contains all of functions, macros, definitions, variables, datatypes, etc. that are required for usage with vendor class function drivers. This file should be included in projects that use vendor class function drivers. This file should also be included into the usb_descriptors.c file and any other user file that requires access to vendor class interfaces.

This file is located in the "<Install Directory>\Microchip\Include\USB" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\Microchip\Include

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

Macros

	Name	Description
~~O	<u>USBGenRead</u>	Receives the specified data out the specified endpoint
0-¢	<u>USBGenWrite</u>	Sends the specified data out the specified endpoint

MCHPFSUSB Device Library > <u>Function Drivers</u> > <u>Vendor Class</u> > <u>Files</u> > <u>usb_function_generic.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

usb_config.h

usb_config.h is a file used to configure the <u>MCHPFSUSB</u> stack and various function drivers. This file provides compile time selection of options provided by the stack. This file defines constants needed by the stack and various function drivers.

This section will detail the definitions required by vendor class function drivers. Additional definitions may be required by the stack itself. Please see the Files topic of the Stack section for more details about stack specific definitions.

Please note that the usb_config.h file can also be generated using the USBConfig.exe tool provided in the "<Install Directory>\USB Tools\USBConfig Tool" directory.

USB_USE_GEN - This define lets the USB stack know that it will be using a vendor class function driver. This should be defined for any device using a vendor class function driver.

#define USB_USE_GEN

USBGEN_EP_SIZE - this defines the size of the IN endpoint used by the MSD driver. The valid range is 1-64 for bulk endpoints and 1-1023 for Isochronous endpoints.

#define USBGEN_EP_SIZE 64

USBGEN_EP_NUM - This defines the endpoint number used by the driver. This number should be unique within the configuration.

#define USBGEN_EP_NUM 1

MCHPFSUSB Device Library > Function Drivers > Vendor Class > Files > usb_config.h

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Stack

Topics

Name	Description
Public API Members	This section includes the Functions, <u>Macros</u> , <u>Variables</u> , Definitions, Constants, and Enumerations that are part of the public API of the <u>MCHPFSUSB</u> stack.
<u>Files</u>	This section lists the files required for use with the device stack. These files should be included in any project using the USB device stack

MCHPFSUSB Device Library > <u>Stack</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Public API Members

This section includes the Functions, <u>Macros</u>, <u>Variables</u>, Definitions, Constants, and Enumerations that are part of the public API of the <u>MCHPFSUSB</u> stack.

Topics

Name	Description
Functions and Macros	
Definitions, Constants, and Enums	
Variables and Types	

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Functions and Macros

Functions

	Name	Description
- = \$	<u>USBDeviceInit</u>	This function initializes the device stack it in the default state. The USB module will be completely reset including all of the internal variables, registers, and interrupt flags.
≡∳	<u>USBDeviceTasks</u>	This function is the main state machine of the USB device side stack. This function should be called periodically to receive and transmit packets through the stack. This function should be called preferably once every 100us during the enumeration process. After the enumeration process this function still needs to be called periodically to respond to various situations on the bus but is more relaxed in its time requirements. This function should also be called at least as fast as the OUT data expected from the PC.
≡ ∳	<u>USBEnableEndpoint</u>	This function will enable the specified endpoint with the specified options
≡∳	<u>USBCBInitEP</u>	This function is called whenever the device receives a SET_CONFIGURATION request.
≡∳	<u>USBCBSuspend</u>	Call back that is invoked when a USB suspend is detected.
≡∳	USBCBWakeFromSuspend	This call back is invoked when a wakeup from USB suspend is detected.
≡Ŵ	<u>USBCBCheckOtherReq</u>	This function is called whenever a request comes over endpoint 0 (the control endpoint) that the stack does not know how to handle.
≡∳	USBCBSendResume	This function should be called to initiate a remote wakeup. (optional)
≡∳	<u>USBCBErrorHandler</u>	This callback is called whenever a USB error occurs. (optional)
≡∳	USBCBStdSetDscHandler	This callback is called when a SET_DESCRIPTOR request is received (optional)

≡ ∳	USBCB_SOF_Handler	This callback is called when a SOF packet is received by the host. (optional)
= \$	USBCBEP0DataReceived	This function is called whenever a EP0 data packet is received. (optional)

Macros

	Name	Description
~~O	<u>USBGetDeviceState</u>	This function will return the current state of the device on the USB. This function should return <u>CONFIGURED_STATE</u> before an application tries to send information on the bus.
~~O	<u>USBGetRemoteWakeupStatus</u>	This function indicates if remote wakeup has been enabled by the host. Devices that support remote wakeup should use this function to determine if it should send a remote wakeup.
~0	USBIsDeviceSuspended	This function indicates if this device is currently suspended. When a device is suspended it will not be able to transfer data over the bus.
~0	<u>USBHandleBusy</u>	Checks to see if the input handle is busy
~0	<u>USBHandleGetAddr</u>	Retrieves the address of the destination buffer of the input handle
~~	<u>USBHandleGetLength</u>	Retrieves the length of the destination buffer of the input handle

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBDeviceInit Function

С

void USBDeviceInit();

Description

This function initializes the device stack it in the default state. The USB module will be completely reset including all of the internal variables, registers, and interrupt flags.

Preconditions

This function must be called before any of the other USB Device functions can be called, including <u>USBDeviceTasks()</u>.

Remarks

None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBDeviceInit Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBDeviceTasks Function

С

```
void USBDeviceTasks();
```

Description

This function is the main state machine of the USB device side stack. This function should be called periodically to receive and transmit packets through the stack. This function should be called preferably once every 100us during the enumeration process. After the enumeration process this function still needs to be called periodically to respond to various situations on the bus but is more relaxed in its time requirements. This function should also be called at least as fast as the OUT data expected from the PC.

Typical usage:

```
Copy Code
void main(void)
{
    USBDeviceInit()
    while(1)
    {
        USBDeviceTasks();
        if((USBGetDeviceState() < CONFIGURED_STATE) ||</pre>
           (<u>USBIsDeviceSuspended()</u> == TRUE))
        {
            //Either the device is not configured or we are suspende
            // so we don't want to do execute any application code
            continue; //go back to the top of the while loop
        }
        else
        {
            //Otherwise we are free to run user application code.
            UserApplication();
        }
    }
}
```

Preconditions

None

Remarks

This function should be called preferably once every 100us during the enumeration process. After the enumeration process this function still needs to be called periodically to respond to various situations on the bus but is more relaxed in its time requirements.

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBDeviceTasks Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBGetDeviceState Macro

С

#define USBGetDeviceState USBDeviceState

Description

This function returns the current state of the device on the USB. This function is used to determine when the device is ready to communicate on the bus. Applications should not try to send or receive data until this function returns <u>CONFIGURED_STATE</u>.

It is also important that applications yield as much time as possible to the <u>USBDeviceTasks</u>() function as possible while the this function returns any value between <u>ATTACHED_STATE</u> through <u>CONFIGURED_STATE</u>.

For more information about the various device states, please refer to the USB specification section 9.1 available from www.usb.org.

Typical usage:

```
Copy Code
void main(void)
{
    USBDeviceInit()
    while(1)
    {
        USBDeviceTasks();
        if((USBGetDeviceState() < CONFIGURED_STATE) ||</pre>
           (USBIsDeviceSuspended() == TRUE))
        {
            //Either the device is not configured or we are suspende
            // so we don't want to do execute any application code
            continue; //go back to the top of the while loop
        }
        else
        {
            //Otherwise we are free to run user application code.
            UserApplication();
        }
    }
```

Preconditions

None

}

Return Values

Return Values	Description
DETACHED_STATE	The device is not attached to the bus
ATTACHED_STATE	The device is attached to the bus but
POWERED_STATE	The device is not officially in the powered state
DEFAULT_STATE	The device has received a RESET from the host
ADR_PENDING_STATE	The device has received the SET_ADDRESS command but hasn't received the STATUS stage of the command so it is still operating on address 0.
ADDRESS_STATE	The device has an address assigned but has not received a SET_CONFIGURATION command yet or has received a SET_CONFIGURATION with a configuration number of 0 (deconfigured)
CONFIGURED_STATE	the device has received a non-zero SET_CONFIGURATION command is now ready for communication on the bus.

Remarks

None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBGetDeviceState Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USBGetRemoteWakeupStatus Macro

С

#define USBGetRemoteWakeupStatus RemoteWakeup

Description

This function indicates if remote wakeup has been enabled by the host. Devices that support remote wakeup should use this function to determine if it should send a remote wakeup.

If a device does not support remote wakeup (the Remote wakeup bit, bit 5, of the bmAttributes field of the Configuration descriptor is set to 1), then it should not send a remote wakeup command to the PC and this function is not of any use to the device. If a device does support remote wakeup then it should use this function as described below.

If this function returns FALSE and the device is suspended, it should not issue a remote wakeup (resume).

If this function returns TRUE and the device is suspended, it should issue a remote wakeup (resume).

A device can add remote wakeup support by having the <u>_RWU</u> symbol added in the configuration descriptor (located in the usb_descriptors.c file in the project). This done in the 8th byte of the configuration descriptor. For example:

Copy Code

```
ROM BYTE configDescriptor1[]={
                                   // Size
  0x09,
  USB_DESCRIPTOR_CONFIGURATION,
                                  // descriptor type
  DESC_CONFIG_WORD(0x0022),
                                   // Total length
                                  // Number of interfaces
  1,
                                  // Index value of this cfg
  1,
                                  // Configuration string index
  0,
  _DEFAULT | _SELF | _RWU,
                                  // Attributes, see usb_device.h
  50,
                                   // Max power consumption in 2X m
```

//The rest of the configuration descriptor should follow

For more information about remote wakeup, see the following section of the USB v2.0 specification available at www.usb.org:

- Section 9.2.5.2
- Table 9-10
- Section 7.1.7.7
- Section 9.4.5

Preconditions

None

Return Values

Return Values	Description
TRUE	Remote Wakeup has been enabled by the host
FALSE	Remote Wakeup is not currently enabled

Remarks

None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBGetRemoteWakeupStatus Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBIsDeviceSuspended Macro

С

#define USBIsDeviceSuspended USBSuspendControl

Description

This function indicates if this device is currently suspended. When a device is suspended it will not be able to transfer data over the bus. This function can be used by the application to skip over section of code that do not need to exectute if the device is unable to send data over the bus.

Typical usage:

```
Copy Code
void main(void)
Ł
    USBDeviceInit()
    while(1)
    {
        USBDeviceTasks();
        if((USBGetDeviceState() < CONFIGURED_STATE) ||</pre>
           (USBIsDeviceSuspended() == TRUE))
        {
            //Either the device is not configured or we are suspe
            // so we don't want to do execute any application co
            continue; //go back to the top of the while loop
        }
        else
        {
            //Otherwise we are free to run user application code.
            UserApplication();
        }
    }
}
```

Preconditions

None

Return Values

Return Values	Description
TRUE	this device is suspended.
FALSE	this device is not suspended.

Remarks

None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBIsDeviceSuspended Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBEnableEndpoint Function

```
C
void USBEnableEndpoint(
BYTE ep,
BYTE options
);
```

void USBCBInitEP(void)

Description

This function will enable the specified endpoint with the specified options.

Typical Usage:

{

}

```
Copy Code
```

USBEnableEndpoint(MSD_DATA_IN_EP,USB_IN_ENABLED|USB_OUT_ENABLED| <u>USBMSDInit</u>();

In the above example endpoint number MSD_DATA_IN_EP is being configured for both IN and OUT traffic with handshaking enabled. Also since MSD_DATA_IN_EP is not endpoint 0 (MSD does not allow this), then we can explicitly disable SETUP packets on this endpoint.

Preconditions

None

Parameters

Parameters	Description
BYTE ep	the endpoint to be configured
	optional settings for the endpoint. The options should be

BYTE options	 ORed together to form a single options string. The available optional settings for the endpoint. The options should be ORed together to form a single options string. The available options are the following: USB_HANDSHAKE_ENABLED enables USB handshaking (ACK, NAK) USB_HANDSHAKE_DISABLED disables USB handshaking (ACK, NAK) USB_OUT_ENABLED enables the out direction USB_OUT_DISABLED disables the out direction USB_IN_ENABLED enables the in direction USB_ALLOW_SETUP enables control transfers USB_DISALLOW_SETUP disables control transfers
--------------	--

Returns

None

Remarks

None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBEnableEndpoint Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBHandleBusy Macro

С

#define USBHandleBusy(handle) (handle==0?0:handle->STAT.UOWN)

Description

Checks to see if the input handle is busy

Typical Usage

Copy Code
//make sure that the last transfer isn't busy by checking the handle
if(!USBHandleBusy(USBGenericInHandle))
{
 //Send the data contained in the INPacket[] array out on
 // endpoint USBGEN_EP_NUM
 USBGenericInHandle = USBGenWrite(USBGEN_EP_NUM,(BYTE*)&INPacket[
}

Preconditions

None

Parameters

Parameters	Description
USB_HANDLE handle	handle of the transfer that you want to check the status of

Return Values

Return Values	Description	
TRUE	The specified handle is busy	
FALSE	The specified handle is free and available for a transfer	

Remarks

None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBHandleBusy Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBHandleGetAddr Macro

С

#define USBHandleGetAddr(handle) (handle->ADR)

Description

Retrieves the address of the destination buffer of the input handle

Preconditions

None

Parameters

Parameters	Description
USB_HANDLE handle	the handle to the transfer you want the address for.

Return Values

Return Values	Description	
WORD	address of the current buffer that the input handle points to.	

Remarks

None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBHandleGetAddr Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBHandleGetLength Macro

С

#define USBHandleGetLength(handle) (handle->CNT)

Description

Retrieves the length of the destination buffer of the input handle

Preconditions

None

Parameters

Parameters	Description
USB_HANDLE handle	the handle to the transfer you want the address for.

Return Values

Return Values	Description	
WORD	length of the current buffer that the input handle points to. If the transfer is complete then this is the length of the data transmitted or the length of data actually received.	

Remarks

None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBHandleGetLength Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBCBInitEP Function

С

```
void USBCBInitEP();
```

Description

This function is called when the device becomes initialized, which occurs after the host sends a SET_CONFIGURATION (wValue not = 0) request. This callback function should initialize the endpoints for the device's usage according to the current configuration.

Typical Usage:

Copy Code

```
void USBCBInitEP(void)
{
    USBEnableEndpoint(MSD_DATA_IN_EP,USB_IN_ENABLED|USB_OUT_ENABLED|
```

```
<u>USBMSDInit();</u>
```

Preconditions

None

}

Remarks

None

```
MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBCBInitEP Function</u>
```

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBCBSuspend Function

С

```
void USBCBSuspend();
```

Description

Call back that is invoked when a USB suspend is detected.

Example power saving code. Insert appropriate code here for the desired application behavior. If the microcontroller will be put to sleep, a process similar to that shown below may be used:

Example Psuedo Code:

```
Copy Code

ConfigureIOPinsForLowPower();

SaveStateOfAllInterruptEnableBits();

DisableAllInterruptEnableBits();

//should enable at least USBActivityIF as a wake source

EnableOnlyTheInterruptsWhichWillBeUsedToWakeTheMicro();

Sleep();

//Preferrably, this should be done in the

// USBCBWakeFromSuspend() function instead.

RestoreStateOfAllPreviouslySavedInterruptEnableBits();

//Preferrably, this should be done in the

// USBCBWakeFromSuspend() function instead.

RestoreIOPinsToNormal();
```

IMPORTANT NOTE: Do not clear the USBActivityIF (ACTVIF) bit here. This bit is cleared inside the <u>usb_device.c</u> file. Clearing USBActivityIF here will cause things to not work as intended.

Preconditions

None

Paramters: None

Side Effects

None

Remark: None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBCBSuspend Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USBCBWakeFromSuspend Function

С

void USBCBWakeFromSuspend();

Description

The host may put USB peripheral devices in low power suspend mode (by "sending" 3+ms of idle). Once in suspend mode, the host may wake the device back up by sending non- idle state signalling.

This call back is invoked when a wakeup from USB suspend is detected.

If clock switching or other power savings measures were taken when executing the <u>USBCBSuspend</u>() function, now would be a good time to switch back to normal full power run mode conditions. The host allows a few milliseconds of wakeup time, after which the device must be fully back to normal, and capable of receiving and processing USB packets. In order to do this, the USB module must receive proper clocking (IE: 48MHz clock must be available to SIE for full speed USB operation).

Preconditions

None

Remarks

None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBCBWakeFromSuspend Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBCBCheckOtherReq Function

С

void USBCBCheckOtherReq();

Description

When SETUP packets arrive from the host, some firmware must process the request and respond appropriately to fulfill the request. Some of the SETUP packets will be for standard USB "chapter 9" (as in, fulfilling chapter 9 of the official USB specifications) requests, while others may be specific to the USB device class that is being implemented. For example, a HID class device needs to be able to respond to "GET REPORT" type of requests. This is not a standard USB chapter 9 request, and therefore not handled by <u>usb_device.c</u>. Instead this request should be handled by class specific firmware, such as that contained in <u>usb_function_hid.c</u>.

Typical Usage:

```
Copy Code
void USBCBCheckOtherReq(void)
{
    //Since the stack didn't handle the request I need to check
    // my class drivers to see if it is for them
    USBCheckMSDRequest();
}
```

Preconditions

None

Remarks

None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBCBCheckOtherReq Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USBCBSendResume Function

С

void USBCBSendResume();

Description

The USB specifications allow some types of USB peripheral devices to wake up a host PC (such as if it is in a low power suspend to RAM state). This can be a very useful feature in some USB applications, such as an Infrared remote control receiver. If a user presses the "power" button on a remote control, it is nice that the IR receiver can detect this signalling, and then send a USB "command" to the PC to wake up.

The USBCBSendResume() "callback" function is used to send this special USB signalling which wakes up the PC. This function may be called by application firmware to wake up the PC. This function should only be called when:

- 1. The USB driver used on the host PC supports the remote wakeup capability.
- 2. The USB configuration descriptor indicates the device is remote wakeup capable in the bmAttributes field. (see usb_descriptors.c and <u>_RWU</u>)
- 3. The USB host PC is currently sleeping, and has previously sent your device a SET FEATURE setup packet which "armed" the remote wakeup capability. (see <u>USBGetRemoteWakeupStatus</u>())

This callback should send a RESUME signal that has the period of 1-15ms.

Typical Usage:

```
if((USBDeviceState == <u>CONFIGURED STATE</u>)
    && (<u>USBIsDeviceSuspended()</u> == TRUE)
    && (<u>USBGetRemoteWakeupStatus()</u> == TRUE))
{
    if(ButtonPressed)
    {
```

Copy Code

```
//Wake up the USB module from suspend
USBWakeFromSuspend();
//Issue a remote wakeup command on the bus
USBCBSendResume();
}
```

Preconditions

None

Remarks

A user can switch to primary first by calling <u>USBCBWakeFromSuspend()</u> if required/desired.

The modifiable section in this routine should be changed to meet the application needs. Current implementation temporary blocks other functions from executing for a period of 1-13 ms depending on the core frequency.

According to USB 2.0 specification section 7.1.7.7, "The remote wakeup device must hold the resume signaling for at lest 1 ms but for no more than 15 ms." The idea here is to use a delay counter loop, using a common value that would work over a wide range of core frequencies. That value selected is 1800. See table below:

Core Freq(MHz)	MIP (for PIC18)	RESUME Signal Period (ms)
48	12	1.05
4	1	12.6

- These timing could be incorrect when using code optimization or extended instruction mode, or when having other interrupts enabled. Make sure to verify using the MPLAB SIM's Stopwatch and verify the actual signal on an oscilloscope.
- These timing numbers should be recalculated when using PIC24 or PIC32 as they have different clocking structures.
- A timer can be used in place of the blocking loop if desired.

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBCBSendResume Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USBCBErrorHandler Function

С

void USBCBErrorHandler();

Description

This callback is called whenever a USB error occurs. (optional)

The purpose of this callback is mainly for debugging during development. Check UEIR to see which error causes the interrupt.

Preconditions

None

Remarks

No need to clear UEIR to 0 here. Callback caller is already doing that.

Typically, user firmware does not need to do anything special if a USB error occurs. For example, if the host sends an OUT packet to your device, but the packet gets corrupted (ex: because of a bad connection, or the user unplugs the USB cable during the transmission) this will typically set one or more USB error interrupt flags. Nothing specific needs to be done however, since the SIE will automatically send a "NAK" packet to the host. In response to this, the host will normally retry to send the packet again, and no data loss occurs. The system will typically recover automatically, without the need for application firmware intervention.

Nevertheless, this callback function is provided, such as for debugging purposes.

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBCBErrorHandler Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USBCBStdSetDscHandler Function

С

void USBCBStdSetDscHandler();

Description

The USBCBStdSetDscHandler() callback function is called when a SETUP, bRequest: SET_DESCRIPTOR request arrives. Typically SET_DESCRIPTOR requests are not used in most applications, and it is optional to support this type of request.

Preconditions

None

Return Values

Return Values	Description
Remark	None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBCBStdSetDscHandler Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USBCB_SOF_Handler Function

С

void USBCB_SOF_Handler();

Description

This callback is called when a SOF packet is received by the host. (optional)

The USB host sends out a SOF packet to full-speed devices every 1 ms. This interrupt may be useful for isochronous pipes. End designers should implement callback routine as necessary.

Preconditions

None

Remarks

None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBCB_SOF_Handler Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USBCBEP0DataReceived Function

С

void USBCBEP0DataReceived();

Description

This function is called whenever a EPO data packet is received. This gives the user (and thus the various class examples a way to get data that is received via the control endpoint. This function needs to be used in conjunction with the <u>USBCBCheckOtherReq</u>() function since the <u>USBCBCheckOtherReq</u>() function is the apps method for getting the initial control transfer before the data arrives.

Preconditions

ENABLE_EP0_DATA_RECEIVED_CALLBACK must be defined already (in usb_config.h)

Remarks

None

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Functions and Macros</u> > <u>USBCBEP0DataReceived Function</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

Definitions, Constants, and Enums

Macros

	Name	Description
v−o	DETACHED_STATE	Detached is the state in which the device is not attached to the bus. When in the detached state a device should not have any pull-ups attached to either the D+ or D- line. This definitions is a return value of the function <u>USBGetDeviceState()</u>
÷	ATTACHED_STATE	Attached is the state in which the device is attached ot the bus but the hub/port that it is attached to is not yet configured. This definitions is a return value of the function <u>USBGetDeviceState()</u>
÷	POWERED_STATE	Powered is the state in which the device is attached to the bus and the hub/port that it is attached to is configured. This definitions is a return value of the function <u>USBGetDeviceState</u> ()
Ŷ	DEFAULT_STATE	Default state is the state after the device receives a RESET command from the host. This definitions is a return value of the function <u>USBGetDeviceState()</u>
v−o	ADR_PENDING_STATE	Address pending state is not an official state of the USB defined states. This state is internally used to indicate that the device has received a SET_ADDRESS command but has not received the STATUS stage of the transfer yet. The device is should not switch addresses until after the STATUS stage is complete. This definitions is a return value of the function USBGetDeviceState()
~	ADDRESS_STATE	Address is the state in which the device has its own specific address on the bus. This definitions is a return value of the function <u>USBGetDeviceState()</u> .
~0	CONFIGURED_STATE	Configured is the state where the device has been fully enumerated and is operating on the bus. The device is now allowed to excute its application specific tasks. It is also allowed to increase its current consumption to the value specified in the configuration descriptor of the current configuration. This definitions is a return value of the function <u>USBGetDeviceState()</u> .

~0~	_DEFAULT	Default Value (Bit 7 is set)
~0	_RWU	Remote Wakeup (Supports if set)
~0	_SELF	Self-powered (Supports if set)

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Definitions, Constants, and Enums</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

DETACHED_STATE Macro

С

#define DETACHED_STATE 0x00

Description

Detached is the state in which the device is not attached to the bus. When in the detached state a device should not have any pull-ups attached to either the D+ or D- line. This definitions is a return value of the function <u>USBGetDeviceState()</u>

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>DETACHED_STATE Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

ATTACHED_STATE Macro

С

#define ATTACHED_STATE 0×01

Description

Attached is the state in which the device is attached of the bus but the hub/port that it is attached to is not yet configured. This definitions is a return value of the function <u>USBGetDeviceState()</u>

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>ATTACHED_STATE Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

POWERED_STATE Macro

С

#define POWERED_STATE 0x02

Description

Powered is the state in which the device is attached to the bus and the hub/port that it is attached to is configured. This definitions is a return value of the function <u>USBGetDeviceState()</u>

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>POWERED_STATE Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

DEFAULT_STATE Macro

С

#define DEFAULT_STATE 0x04

Description

Default state is the state after the device receives a RESET command from the host. This definitions is a return value of the function <u>USBGetDeviceState()</u>

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>DEFAULT_STATE Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

ADR_PENDING_STATE Macro

С

#define ADR_PENDING_STATE 0x08

Description

Address pending state is not an official state of the USB defined states. This state is internally used to indicate that the device has received a SET_ADDRESS command but has not received the STATUS stage of the transfer yet. The device is should not switch addresses until after the STATUS stage is complete. This definitions is a return value of the function <u>USBGetDeviceState()</u>

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>ADR_PENDING_STATE Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

ADDRESS_STATE Macro

С

#define ADDRESS_STATE 0x10

Description

Address is the state in which the device has its own specific address on the bus. This definitions is a return value of the function <u>USBGetDeviceState()</u>.

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>ADDRESS_STATE Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

CONFIGURED_STATE Macro

С

#define CONFIGURED_STATE 0x20

Description

Configured is the state where the device has been fully enumerated and is operating on the bus. The device is now allowed to excute its application specific tasks. It is also allowed to increase its current consumption to the value specified in the configuration descriptor of the current configuration. This definitions is a return value of the function <u>USBGetDeviceState()</u>.

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>CONFIGURED_STATE Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

USB Device Library Help	Contents Index Home	Previous Up Next
_DEFAULT Macro		
С		
<pre>#define _DEFAULT (0x01<<7</pre>) //Default Value	(Bit 7 is set)
2		

Description

Default Value (Bit 7 is set)

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>_DEFAULT Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

RWU Macro

С

#define _RWU (0x01<<5)</pre>

//Remote Wakeup (Supports if set)

Description

Remote Wakeup (Supports if set)

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>RWU Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

#define _SELF (0x01<<6)</pre>

//Self-powered (Supports if set)

Description

Self-powered (Supports if set)

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Definitions, Constants, and Enums</u> > <u>_SELF Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Variables and Types

Macros

	Name	Description
~~	USB_HANDLE	USB_HANDLE is a pointer to an entry in the BDT. This pointer can be used to read the length of the last transfer, the status of the last transfer, and various other information. Insure to initialize USB_HANDLE objects to NULL so that they are in a known state during their first usage.

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Variables and Types</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

USB_HANDLE Macro

С

#define USB_HANDLE volatile BDT_ENTRY*

Description

USB_HANDLE is a pointer to an entry in the BDT. This pointer can be used to read the length of the last transfer, the status of the last transfer, and various other information. Insure to initialize USB_HANDLE objects to NULL so that they are in a known state during their first usage.

MCHPFSUSB Device Library > <u>Stack</u> > <u>Public API Members</u> > <u>Variables and Types</u> > <u>USB_HANDLE Macro</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Files

This section lists the files required for use with the device stack. These files should be included in any project using the USB device stack

Files

Name	Description
usb_device.c	This file contains functions, macros, definitions, variables, datatypes, etc. that are required for usage with the <u>MCHPFSUSB</u> device stack. This file should be included in projects that use the device stack. This file is located in the " <install Directory>\Microchip\USB" directory.</install
<u>usb.h</u>	This file aggregates all necessary header files for the Microchip USB Host, Device, and OTG libraries. It provides a single-file can be included in application code. The USB libraries simplify the implementation of USB applications by providing an abstraction of the USB module and its registers and bits such that the source code for the can be the same across various hardware platforms.
usb_ch9.h	This file defines data structures, constants, and macros that are used to to support the USB Device Framework protocol described in Chapter 9 of the USB 2.0 specification.
usb_common.h	This file defines data types, constants, and macros that are common to multiple layers of the Microchip USB Firmware Stack.
usb_device.h	This file, with its associated C source file, provides the main substance of the USB device side stack. These files will receive, transmit, and process various USB commands as well as take action when required for various events that occur on the bus.
usb_hal.h	This file abstracts the hardware interface.
usb_hal_pic18.h	This file abstracts the hardware interface. The USB stack firmware can be compiled to work on different USB microcontrollers, such as PIC18 and PIC24. The USB related special function registers and bit names are generally very similar between the device families, but

	small differences in naming exist.
<u>usb_hal_pic24.h</u>	This file abstracts the hardware interface. The USB stack firmware can be compiled to work on different USB microcontrollers, such as PIC18 and PIC24. The USB related special function registers and bit names are generally very similar between the device families, but small differences in naming exist.
usb_hal_pic32.h	This file abstracts the hardware interface. The USB stack firmware can be compiled to work on different USB microcontrollers, such as PIC18 and PIC24. The USB related special function registers and bit names are generally very similar between the device families, but small differences in naming exist.

Topics

Name	Description
usb_config.h	usb_config.h is a file used to configure the <u>MCHPFSUSB</u> stack. This file provides compile time selection of options provided by the stack. This file defines constants needed by the stack and various function drivers.
HardwareProfile.h	HardwareProfile.h is a file used to define hardware specific definitions that are required by the <u>MCHPFSUSB</u> stack. This file should be modified to match the application hardware.

MCHPFSUSB Device Library > <u>Stack</u> > <u>Files</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

usb_device.c

USB Device Stack File

This file contains functions, macros, definitions, variables, datatypes, etc. that are required for usage with the <u>MCHPFSUSB</u> device stack. This file should be included in projects that use the device stack.

This file is located in the "<Install Directory>\Microchip\USB" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\MicrochipInclude

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

MCHPFSUSB Device Library > <u>Stack</u> > <u>Files</u> > <u>usb_device.c</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

usb.h

USB Header File

This file aggregates all necessary header files for the Microchip USB Host, Device, and OTG libraries. It provides a single-file can be included in application code. The USB libraries simplify the implementation of USB applications by providing an abstraction of the USB module and its registers and bits such that the source code for the can be the same across various hardware platforms.

Note that this file does not include the header files for any client or function drivers.

This file is located in the "<Install Directory>\Microchip\Include\USB" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\MicrochipInclude

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

MCHPFSUSB Device Library > <u>Stack</u> > <u>Files</u> > <u>usb.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

usb_ch9.h

USB Chapter 9 Protocol (Header File)

This file defines data structures, constants, and macros that are used to to support the USB Device Framework protocol described in Chapter 9 of the USB 2.0 specification.

This file is located in the "<Install Directory>\Microchip\Include\USB" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\MicrochipInclude

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

MCHPFSUSB Device Library > <u>Stack</u> > <u>Files</u> > <u>usb_ch9.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

usb_common.h

Common USB Library Definitions (Header File)

This file defines data types, constants, and macros that are common to multiple layers of the Microchip USB Firmware Stack.

This file is located in the "<Install Directory>\Microchip\Include\USB" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\MicrochipInclude

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

MCHPFSUSB Device Library > <u>Stack</u> > <u>Files</u> > <u>usb_common.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

usb_device.h

USB Device header file

This file, with its associated C source file, provides the main substance of the USB device side stack. These files will receive, transmit, and process various USB commands as well as take action when required for various events that occur on the bus.

This file is located in the "<Install Directory>\Microchip\Include\USB" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\MicrochipInclude

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

Functions

	Name	Description
= \$	USBCB_SOF_Handler	This callback is called when a SOF packet is received by the host. (optional)
≡∳	<u>USBCBCheckOtherReq</u>	This function is called whenever a request comes over endpoint 0 (the control endpoint) that the stack does not know how to handle.
-= \$	USBCBEP0DataReceived	This function is called whenever a EP0 data packet is received. (optional)
-= \$	<u>USBCBErrorHandler</u>	This callback is called whenever a USB error occurs. (optional)
-= \$	<u>USBCBInitEP</u>	This function is called whenever the device receives a SET_CONFIGURATION request.
-= \$	<u>USBCBSendResume</u>	This function should be called to initiate a remote wakeup. (optional)
	USBCBStdSetDscHandler	This callback is called when a SET_DESCRIPTOR request is received (optional)
	USBCBSuspend	Call back that is invoked when a USB suspend is detected.
	USBCBWakeFromSuspend	This call back is invoked when a wakeup from USB suspend is detected.
- :	<u>USBDeviceInit</u>	This function initializes the device stack it in the default state. The USB module will be completely reset including all of the internal variables, registers, and interrupt flags.
=•	<u>USBDeviceTasks</u>	This function is the main state machine of the USB device side stack. This function should be called periodically to receive and transmit packets through the stack. This function should be called preferably once every 100us during the enumeration process. After the enumeration process this function still needs to be called periodically to respond to various situations on the bus but is more relaxed in its time requirements. This function should also be called at least as fast as the OUT data expected from the PC.

- =		This function will enable the specified endpoint with the specified options
------------	--	---

Macros

	Name	Description
~~O	_DEFAULT	Default Value (Bit 7 is set)
~0	_RWU	Remote Wakeup (Supports if set)
÷	_SELF	Self-powered (Supports if set)
→ 0	ADDRESS_STATE	Address is the state in which the device has its own specific address on the bus. This definitions is a return value of the function <u>USBGetDeviceState</u> ().
O	ADR_PENDING_STATE	Address pending state is not an official state of the USB defined states. This state is internally used to indicate that the device has received a SET_ADDRESS command but has not received the STATUS stage of the transfer yet. The device is should not switch addresses until after the STATUS stage is complete. This definitions is a return value of the function <u>USBGetDeviceState(</u>)
~0	ATTACHED_STATE	Attached is the state in which the device is attached ot the bus but the hub/port that it is attached to is not yet configured. This definitions is a return value of the function <u>USBGetDeviceState(</u>)
O	CONFIGURED_STATE	Configured is the state where the device has been fully enumerated and is operating on the bus. The device is now allowed to excute its application specific tasks. It is also allowed to increase its current consumption to the value specified in the configuration descriptor of the current configuration. This definitions is a return value of the function <u>USBGetDeviceState()</u> .
~~	DEFAULT_STATE	Default state is the state after the device receives a RESET command from the host. This definitons is a return value of the function <u>USBGetDeviceState()</u>
		Detached is the state in which the device is not

~~O	DETACHED_STATE	attached to the bus. When in the detached state a device should not have any pull-ups attached to either the D+ or D- line. This definitions is a return value of the function <u>USBGetDeviceState(</u>)
~~0	POWERED_STATE	Powered is the state in which the device is attached to the bus and the hub/port that it is attached to is configured. This defintions is a return value of the function <u>USBGetDeviceState()</u>
~~	USB_HANDLE	USB_HANDLE is a pointer to an entry in the BDT. This pointer can be used to read the length of the last transfer, the status of the last transfer, and various other information. Insure to initialize USB_HANDLE objects to NULL so that they are in a known state during their first usage.
~~	<u>USBGetDeviceState</u>	This function will return the current state of the device on the USB. This function should return <u>CONFIGURED_STATE</u> before an application tries to send information on the bus.
~~	<u>USBGetRemoteWakeupStatus</u>	This function indicates if remote wakeup has been enabled by the host. Devices that support remote wakeup should use this function to determine if it should send a remote wakeup.
~~	<u>USBHandleBusy</u>	Checks to see if the input handle is busy
~	<u>USBHandleGetAddr</u>	Retrieves the address of the destination buffer of the input handle
~~	<u>USBHandleGetLength</u>	Retrieves the length of the destination buffer of the input handle
~~0	<u>USBIsDeviceSuspended</u>	This function indicates if this device is currently suspended. When a device is suspended it will not be able to transfer data over the bus.

MCHPFSUSB Device Library > <u>Stack</u> > <u>Files</u> > <u>usb_device.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

usb_hal.h

USB Hardware Abstraction Layer (HAL) (Header File)

This file abstracts the hardware interface.

This file is located in the "<Install Directory>\Microchip\Include\USB" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\MicrochipInclude

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

MCHPFSUSB Device Library > <u>Stack</u> > <u>Files</u> > <u>usb_hal.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008

Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

Previous | Up | Next

usb_hal_pic18.h

USB Hardware Abstraction Layer (HAL) (Header File)

This file abstracts the hardware interface. The USB stack firmware can be compiled to work on different USB microcontrollers, such as PIC18 and PIC24. The USB related special function registers and bit names are generally very similar between the device families, but small differences in naming exist.

In order to make the same set of firmware work accross the device families, when modifying SFR contents, a slightly abstracted name is used, which is then "mapped" to the appropriate real name in the usb_hal_picxx.h header.

Make sure to include the correct version of the usb_hal_picxx.h file for the microcontroller family which will be used.

This file is located in the "<Install Directory>\Microchip\Include\USB" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\MicrochipInclude

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

MCHPFSUSB Device Library > <u>Stack</u> > <u>Files</u> > <u>usb_hal_pic18.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USB Device Library Help

Contents | Index | Home

Previous | Up | Next

usb_hal_pic24.h

USB Hardware Abstraction Layer (HAL) (Header File)

This file abstracts the hardware interface. The USB stack firmware can be compiled to work on different USB microcontrollers, such as PIC18 and PIC24. The USB related special function registers and bit names are generally very similar between the device families, but small differences in naming exist.

In order to make the same set of firmware work accross the device families, when modifying SFR contents, a slightly abstracted name is used, which is then "mapped" to the appropriate real name in the usb_hal_picxx.h header.

Make sure to include the correct version of the usb_hal_picxx.h file for the microcontroller family which will be used.

This file is located in the "<Install Directory>\Microchip\Include\USB" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\MicrochipInclude

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

MCHPFSUSB Device Library > <u>Stack</u> > <u>Files</u> > <u>usb_hal_pic24.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\ensuremath{\mathbb{O}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USB Device Library Help

Contents | Index | Home

Previous | Up | Next

usb_hal_pic32.h

USB Hardware Abstraction Layer (HAL) (Header File)

This file abstracts the hardware interface. The USB stack firmware can be compiled to work on different USB microcontrollers, such as PIC18 and PIC24. The USB related special function registers and bit names are generally very similar between the device families, but small differences in naming exist.

In order to make the same set of firmware work accross the device families, when modifying SFR contents, a slightly abstracted name is used, which is then "mapped" to the appropriate real name in the usb_hal_picxx.h header.

Make sure to include the correct version of the usb_hal_picxx.h file for the microcontroller family which will be used.

This file is located in the "<Install Directory>\Microchip\Include\USB" directory.

When including this file in a new project, this file can either be referenced from the directory in which it was installed or copied directly into the user application folder. If the first method is chosen to keep the file located in the folder in which it is installed then include paths need to be added so that the library and the application both know where to reference each others files. If the application folder is located in the same folder as the Microchip folder (like the current demo folders), then the following include paths need to be added to the application's project:

..\Include

..\..\Include

..\..\MicrochipInclude

..\..\<Application Folder>

..\..\<Application Folder>

If a different directory structure is used, modify the paths as required. An example using absolute paths instead of relative paths would be the following:

C:\Microchip Solutions\Microchip\Include

C:\Microchip Solutions\My Demo Application

MCHPFSUSB Device Library > <u>Stack</u> > <u>Files</u> > <u>usb_hal_pic32.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright @ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USB Device Library Help

Contents | Index | Home

usb_config.h

usb_config.h is a file used to configure the <u>MCHPFSUSB</u> stack. This file provides compile time selection of options provided by the stack. This file defines constants needed by the stack and various function drivers.

This section will detail the definitions required by the stack. Additional definitions may be required by the function drivers. Please see the Files topic of the Function Drivers section for more details about function driver specific definitions.

Please note that the usb_config.h file can also be generated using the USBConfig.exe tool provided in the "<Install Directory>\USB Tools\USBConfig Tool" directory.

USB_USE_DEVICE - This enables the stack to run in device mode and must be included in any project using the USB device side firmware.

#define USB_USE_DEVICE

USB_EP0_BUFF_SIZE - this defines the size of the endpoint 0 buffer. Using larger options take more SRAM, but does not provide much advantage in most types of applications. Exceptions to this, are applications that use EP0 IN or OUT for sending large amounts of application related data. The available options are 8, 16, 32, or 64.

#define USB_EP0_BUFF_SIZE 8 // Valid Options: 8, 16, 32, or 64 bytes.

USB_MAX_NUM_INT - this defines the maximum interface number used by the device. If the device uses multiple configurations then this is the largest interface number in any of the configurations.

#define USB_MAX_NUM_INT 1 // For tracking Alternate Setting

USB_USER_DEVICE_DESCRIPTOR - this defines a pointer to the device descriptor. If this definition is not defined then the application must define a ROM variable named device_dsc that holds the device descriptor.

#define USB_USER_DEVICE_DESCRIPTOR &device_dsc

USB_USER_DEVICE_DESCRIPTOR_INCLUDE - this defines any prototype that needs to be included into the library files in order to access the variables/functions required by the USB_USER_DEVICE_DESCRIPTOR option.

Example where USB_USER_DEVICE_DESCRIPTOR is a ROM variable:

#define USB_USER_DEVICE_DESCRIPTOR &device_dsc

#define USB_USER_DEVICE_DESCRIPTOR_INCLUDE extern ROM
USB_DEVICE_DESCRIPTOR device_dsc

Example where USB_USER_DEVICE_DESCRIPTOR is a function:

#define USB_USER_DEVICE_DESCRIPTOR myDeviceDescriptorFunction()

#define USB_USER_DEVICE_DESCRIPTOR_INCLUDE ROM BYTE*
myDeviceDescriptorFunction(void)

USB_USER_CONFIG_DESCRIPTOR - this defines a pointer to the configuration descriptor. If this definition is not defined then the application must define a ROM variable named USB_CD_Ptr that holds the configuration descriptor.

USB_USER_CONFIG_DESCRIPTOR_INCLUDE - this defines any prototype that needs to be included into the library files in order to access the variables/functions required by the USB_USER_DEVICE_DESCRIPTOR option.

Example where USB_USER_CONFIG_DESCRIPTOR is a ROM variable:

#define USB_USER_CONFIG_DESCRIPTOR USB_CD_Ptr

#define USB_USER_CONFIG_DESCRIPTOR_INCLUDE extern ROM BYTE
USB_CD_Ptr[]

Example where USB_USER_CONFIG_DESCRIPTOR is a function:

#define USB_USER_CONFIG_DESCRIPTOR myConfigDescriptorFunction()

#define USB_USER_CONFIG_DESCRIPTOR_INCLUDE ROM BYTE*
myConfigDescriptorFunction(void)

USB_PING_PONG_MODE - This defines the ping pong mode of operation. When ping ponging is enabled, the device is capable of preparing up to two buffers to send/receive per endpoint direction at any given point of time. This can be used to increase throughput but requires additional RAM. USB_PING_PONG_MODE must be defined as one of the following options:

- USB_PING_PONG__NO_PING_PONG
- USB_PING_PONG__FULL_PING_PONG
- USB_PING_PONG__EP0_OUT_ONLY
- USB_PING_PONG__ALL_BUT_EP0

Example:

#define USB_PING_PONG_MODE USB_PING_PONG__FULL_PING_PONG

Please note that not every device can support all ping pong modes. Please refer to the device datasheet for available options. Here are some limitations:

• PIC18F4550 family rev A3 devices do not support USB_PING_PONG__ALL_BUT_EP0

• PIC32MX460F512L family devices only support USB_PING_PONG__FULL_PING_PONG USB_POLLING - this definition tells the stack that it is going to be polled. The user is responsible for calling <u>USBDeviceTasks</u>() frequently enough to keep the USB stack running. Currently this is the only available option.

#define USB_POLLING

USB_PULLUP_OPTION - This option tells the USB stack if it should enable the internal pull-up resistors or not. There are two possible options:

- USB_PULLUP_ENABLE
- USB_PULLUP_DISABLE Example:

#define USB_PULLUP_OPTION USB_PULLUP_ENABLE

USB_TRANSCEIVER_OPTION - This option allows the user to specify if they wish to use the internal or external USB transceiver module. The available options are:

- USB_INTERNAL_TRANSCEIVER
- USB_EXTERNAL_TRANSCEIVER

Example:

#define USB_TRANSCEIVER_OPTION USB_INTERNAL_TRANSCEIVER

USB_SPEED_OPTION - This option allows users to select the USB speed that want to operate. The available options are the following:

- USB_FULL_SPEED
- USB_LOW_SPEED

Example:

#define USB_SPEED_OPTION USB_FULL_SPEED

Please note that not all options are available on every device. Please refer to the appropriate device datasheet for more information about a devices capabilities. Some of the known limitations are the following:

- PIC24FJ256GB110 family devices only support USB_FULL_SPEED
- PIC32MX460F512L family devices only support USB_FULL_SPEED

MCHPFSUSB Device Library > <u>Stack</u> > <u>Files</u> > <u>usb_config.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

HardwareProfile.h

HardwareProfile.h is a file used to define hardware specific definitions that are required by the <u>MCHPFSUSB</u> stack. This file should be modified to match the application hardware.

The following definitions are used by the stack and example code. Additional definitions may be required by the function drivers used in the application. Please see the HardwareProfiles.h Topic in each of the function driver sections for additional information about class specific definition requirements.

GetSystemClock() - this macro should be defined to return the current clock rate of the CPU. Because this rate is defined as the rate the CPU is clocked then any PLLs, pre-scalars, post-scalars, etc. should already be included in this value.

#define GetSystemClock() 48000000

PROGRAMMABLE_WITH_USB_HID_BOOTLOADER - The existence of this definition tells the example firmware that this application will be bootloaded using the HID bootloader. The example applications will remap any vectors required to the appropriate address if this is defined. This definition is optional.

#define PROGRAMMABLE_WITH_USB_HID_BOOTLOADER

PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER - The existence of this definition tells the example firmware that this application will be bootloaded using the MCHPUSB bootloader. The example applications will remap any vectors required to the appropriate address if this is defined. The MCHPUSB bootloader only supports the PIC18F4550 and PIC18F4450 device families. This definition is optional.

#define PROGRAMMABLE_WITH_USB_MCHPUSB_BOOTLOADER

USE_SELF_POWER_SENSE_IO - The existence of this definition tells the stack that the application is capable of sensing if it is self powered

or not. If this is defined then the tris_self_power and self_power definitions must also be defined.

```
#define USE_SELF_POWER_SENSE_I0
```

tris_self_power - This definition tells the firmware which TRIS pin to tri-state in order to detect if the device is self powered or not. If the self power is determined from something other than a port pin, then this definition can point to a dummy variable bit.

```
#define tris_self_power TRISAbits.TRISA2 // Input
```

self_power - This definition is what is used by the stack to determine if the device is currently self powered. This can be a function or a bit. This definition should equate to 1 if the device is self powered and 0 if the device is not currently self powered. This definition should exist irrespective of if the device is capable of sensing its own self power.

```
#if defined(USE_SELF_POWER_SENSE_IO)
#define self_power PORTAbits.RA2
#else
#define self_power 1
```

#endif

USE_USB_BUS_SENSE_IO - The existence of this definition tells the stack that the application is capable of sensing if it is bus powered or not. If this is defined then the tris_usb_bus_sense and USB_BUS_SENSE definitions must also be defined.

#define USE_USB_BUS_SENSE_I0

tris_self_power - This definition tells the firmware which TRIS pin to tri-state in order to detect if the device is bus powered or not. If the bus power is determined from something other than a port pin, then this definition can point to a dummy variable bit.

#define tris_usb_bus_sense TRISAbits.TRISA1 // Input

USB_BUS_SENSE - This definition is what is used by the stack to determine if the device is currently self powered. This can be a function or a bit. This definition should equate to 1 if the device is self powered and 0 if the device is not currently self powered. This definition should exist irrespective of if the device is capable of sensing its own self power.

#if defined(USE_USB_BUS_SENSE_IO)

#define USB_BUS_SENSE PORTAbits.RA1

#else

#define USB_BUS_SENSE 1

#endif

MCHPFSUSB Device Library > <u>Stack</u> > <u>Files</u> > <u>HardwareProfile.h</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USB Device Library Help

Contents | Index | Home

Previous | Next

MCHPFSUSB

License Agreement for usage of the MCHPFSUSB v2.3 library.

Remarks

IMPORTANT:

MICROCHIP TECHNOLOGY INC. ("COMPANY") IS WILLING TO LICENSE USB FRAMEWORK SOFTWARE AND ACCOMPANYING DOCUMENTATION OFFERED TO YOU ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE FOLLOWING TERMS. TO ACCEPT THE TERMS OF THIS LICENSE, CLICK "I ACCEPT" AND PROCEED WITH THE DOWNLOAD OR INSTALL. IF YOU DO NOT ACCEPT THESE LICENSE TERMS, CLICK "I DO NOT ACCEPT," AND DO NOT DOWNLOAD OR INSTALL THIS SOFTWARE.

NON-EXCLUSIVE SOFTWARE LICENSE AGREEMENT

FOR MICROCHIP USB FRAMEWORK SOFTWARE

IMPORTANT - READ CAREFULLY.

This Nonexclusive Software License Agreement ("Agreement") is a contract between you, either as an individual or a single entity, ("Licensee") and Microchip Technology Incorporated ("Company") for Company's USB Framework software which may include firmware source code ("Software") and accompanying proprietary documentation ("Documentation").

The Software and Documentation are licensed under this Agreement and not sold. The Software and Documentation are protected by U.S. copyright laws and international copyright treaties, and other intellectual property laws and treaties.

In consideration of the mutual covenants of the parties and for the consideration received herein, the parties hereby agree as follows:

1. Ownership of Proprietary Rights. Except as expressly licensed herein, Company retains all right, title and interest in and to the Software ("Proprietary Rights"), including, but not limited to: (i) patent, copyright, trade secret and similar right in the Software and underlying technology; (ii) all copies and derivative works thereof (by whomever produced) and (iii) the Documentation. Licensee shall have no right to modify, use, copy, license, create derivative works of, sell, assign or otherwise transfer all or any portion of the Software or the Proprietary Rights for the Software except as expressly set forth in the Agreement. Except to the extent provided in this Agreement, all uses of the Proprietary Rights of the Software hereunder, including hardware, firmware and software implementations of the technology, will inure to the benefit of the Company, and any all equities or rights in and to the Proprietary Rights of the Software resulting from Licensee's acts or endeavors hereunder will automatically transfer to the Company.

2. License Grant.

(a) Subject to the terms of this Agreement, Company grants, strictly to Licensee, a personal, worldwide, fully paid-up, non-exclusive, nontransferable, limited license to reproduce, market, license, distribute, use and create derivative works of the Software, solely for incorporation into a product manufactured by Licensee which only implements the Software on proprietary products manufactured by Company and purchased by Licensee from Company or Company's authorized distributor (a "Device"). The product produced by the Licensee which integrates a Device programmed with the Software shall hereafter be called a "Licensee Product." The Company also grants, strictly to Licensee, a personal, worldwide, fully paid-up, non-exclusive, nontransferable, limited license to use the Documentation in support of Licensee's authorized use of the Software.

(b) Subject to the terms of this Agreement, Licensee may grant a limited sublicense to a third party ("Sublicensee") of the rights granted to Licensee in Section 2(a) above with the following additional limitations. Such sublicense shall permit the Sublicensee to use the Software only as described below in subparagraphs 2(b)(i) through 2(b) (v) and only IF such Sublicensee expressly agrees in writing to be bound by terms of confidentiality and limited use that are no broader in scope and duration than the confidentiality and limited use terms of this Agreement:

(i) Sublicensee may modify source code solely for use by Licensee.

(ii) Sublicensee may program Software into Devices for Licensee.

(iii) Sublicensee may use the Software to develop and/or manufacture Licensee Products.

(iv) Sublicensee may use the Software to develop and/or manufacture products where either: (x) the sublicensed Software contains source code modified or otherwise optimized by Licensee for Sublicensee use and the Software is implemented only on proprietary products manufactured by Company and purchased from Company or Company's authorized distributor; or (y) the sublicensed Software is programmed into a Device by Licensee on behalf of such Sublicensee.

(v) Sublicensee may use the Documentation in support of Sublicensee's authorized use of the Software in conformance with this Section 2(b). Sublicensee shall have no right to further sublicense any right granted under this Agreement.

(c) Authorized representatives of the Company shall have the right to reasonably inspect Licensee's or Sublicensee's premises and to audit Licensee's or Sublicensee's records and inventory of products in which the Software is implemented, whether located on Licensee's or Sublicensee's premises or elsewhere at any time, announced or unannounced, and in its sole and absolute discretion, in order to ensure Licensee's and Sublicensee's adherence to the terms of this Agreement.

3. Licensee Obligations.

(a) Licensee will not (i) engage in unauthorized sublicensing, disclosure or distribution of Software or Documentation, (ii) use all or any portion of the Software or the Proprietary Rights of the Software except in conjunction with a Device; or (iii) reverse engineer (by disassembly, decompilation or otherwise) the Software. For purposes of clarity, Licensee's use of the Software binary driver file on a Windows® operating system to interface with a Device does not violate this Section 3(a).

(b) Licensee will not, during the term of this Agreement or any time thereafter, attack, dispute or contest, directly or indirectly, the Company's exclusive right, title and interest in or to the Proprietary Rights of the Software or the validity of the Company's ownership thereof, nor will Licensee assist or encourage others to do so.

4. Indemnification.

(a) Licensee Indemnity. Licensee will defend, indemnify and hold Company harmless from and against any and all claims, losses, liabilities, damages, costs, and expenses (including attorney's fees) directly or indirectly arising from or related to (i) any use or disclosure of the Software not permitted under this Agreement, (ii) any use of the Software in combination with other products, equipment, software or data not supplied by Company, including but not limited to use in the Licensee Product, (iii) any modification of the Software made by any person other than Company, or (iv) any products liability claims with respect to the Licensee Product.

(b) THE FOREGOING STATES THE SOLE AND EXCLUSIVE REMEDY AND LIABILITY OF THE PARTIES FOR INTELLECTUAL PROPERTY INFRINGEMENT.

5. Term and Termination.

(a) Term. This Agreement and the license granted herein shall be effective from the date of acceptance of the Agreement, or first installation, copy or use of the Software, whichever event occurs first. The term of this Agreement shall run until terminated in accordance with this Section 5.

(b) Termination for Cause. This Agreement may be terminated if either party materially fails to perform or comply with any provisions of this Agreement. In the event of Licensee's breach, termination shall be effective immediately without notice from Company.

(c) Termination for Convenience. This Agreement may be terminated by

either party for convenience upon 60 days written notice.

(d) Effects of Termination. Upon termination, Licensee shall immediately discontinue all use of the Software, including but limited to inclusion in any Licensee Product, and shall destroy any tangible media Licensee has on which the Software or Documentation exist, and remove the Software and Documentation from any and all systems.

6. Injunctive Relief. Licensee agrees that the provisions in this Agreement regarding unauthorized use of the Software and nondisclosure are necessary to protect the legitimate business interests of the Company. Licensee also agrees that monetary damages alone cannot adequately compensate the Company if there is a violation of such provisions by Licensee and that injunctive relief against Licensee is essential for the protection of the Company. Licensee agrees, therefore, that if the Company alleges that Licensee has breached or violated such provisions then, in addition to any other remedies it may have, the Company will have the right to petition a court of competent jurisdiction, with the requirement for the posting of a bond, for injunctive relief against Licensee in addition to all other remedies at law or in equity.

7. Confidentiality. Licensee agrees that all source code, source documentation and underlying inventions, algorithms, know-how and ideas relating to the Software and the Documentation are the Company's proprietary information ("Proprietary Information"). Except as expressly and unambiguously allowed herein, the Licensee will hold in confidence and not use or disclose any Proprietary Information and shall similarly bind its employees and Subcontractors in writing. Proprietary Information shall not include information that (i) is in or enters the public domain without breach of this Agreement and through no fault of the receiving party; (ii) the receiving party was legally in possession of prior to receiving it; (iii) the receiving party can demonstrate was developed by it independently and without use of or reference to the disclosing party's Proprietary Information; or (iv) the receiving party receives from a third party without restriction on disclosure. If a party is required to disclose Proprietary Information by law, court order, or government agency, such disclosure shall not be deemed a breach of this Agreement. Licensee's obligation under this Section 7 will survive for a period of twenty (20) years following the expiration or termination of this Agreement.

8. Warranties and Disclaimers. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE. LICENSEE ASSUMES THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE AND DOCUMENTATION, AND COMPANY ASSUMES NO RESPONSIBILITY FOR THE ACCURACY OR APPLICATION OR OF ERRORS OR OMISSIONS IN THE SOFTWARE. THE LIMITED REMEDIES SET FORTH HEREIN SHALL APPLY NOTWITHSTANDING FAILURE OF THEIR ESSENTIAL PURPOSE.

10. Limited Liability. IN NO EVENT SHALL COMPANY BE LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL OR EQUITABLE THEORY FOR ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS. The aggregate and cumulative liability of Company for damages hereunder will in no event exceed \$100 and Licensee acknowledges that the foregoing limitations are reasonable and an essential part of this Agreement.

11. General.

(a) Electronic Notices and Requests. When Licensee visits Company's website or send e-mails to the Company, Licensee ise communicating with Company electronically. Licensee hereby consents to receive communications from Company electronically. Company will communicate with Licensee by e-mail or by posting notices on www.Microchip.com. Licensee agrees that all agreements, notices, disclosures and other communications that Company provide to Licensee electronically satisfy any legal requirement that such communications be in writing. In giving notice to Company, Licensee will address notices to Legal.Department@Microchip.com. and include Licensee's name, company name, physical address and phone number, and include "USB Framework Software" in the subject line.

(b) Governing Law. THIS AGREEMENT SHALL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF THE STATE OF ARIZONA AND THE UNITED STATES WITHOUT REGARD TO CONFLICTS OF LAWS PROVISIONS. THEREOF AND WITHOUT REGARD TO THE UNITED NATIONS CONVENTION ON CONTRACTS FOR THE INTERNATIONAL SALE OF GOODS. The sole jurisdiction and venue for actions related to the subject matter hereof shall be the state and federal courts located in Arizona.

(c) Attorneys' Fees. If either the Company or Licensee employs attorneys to enforce any rights arising out of or relating to this Agreement, the prevailing party shall be entitled to recover its reasonable attorneys' fees, costs and other expenses.

(d) Entire Agreement. This Agreement shall constitute the entire agreement between the parties with respect to the subject matter hereof. It shall not be modified except by a written agreement signed by an authorized representative of the Company.

(e) Severability. If any provision of this Agreement shall be held by a court of competent jurisdiction to be illegal, invalid or unenforceable, that provision shall be limited or eliminated to the minimum extent necessary so that this Agreement shall otherwise remain in full force and effect and enforceable.

(f) Waiver. No waiver of any breach of any provision of this Agreement shall constitute a waiver of any prior, concurrent or subsequent breach of the same or any other provisions hereof, and no waiver shall be effective unless made in writing and signed by an authorized representative of the waiving party.

(g) Export Regulation. Each party hereby agrees to comply with all export laws and restrictions and regulations of the Department of Commerce or other United States or foreign agency or authority.

(h) Survival. The indemnities herein, and any right of action for breach of this Agreement prior to termination shall survive any termination of this Agreement.

(i) Assignment. Neither this Agreement nor any rights, licenses or obligations hereunder, may be assigned by Licensee without the prior written approval of the Company.

License Rev. 02-032907

License Agreements > <u>MCHPFSUSB</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USB Device Library Help

Contents | Index | Home

Previous | Next

MDD File System

License Agreement for usage of the Microchip MDD File System v1.2

Remarks

IMPORTANT:

MICROCHIP IS WILLING TO LICENSE THE MEMORY DISK DRIVE FILE SYSTEM SOFTWARE AND ACCOMPANYING DOCUMENTATION OFFERED TO YOU ONLY ON THE CONDITION THAT YOU ACCEPT ALL OF THE FOLLOWING TERMS. TO ACCEPT THE TERMS OF THIS LICENSE, CLICK "I ACCEPT" AND PROCEED WITH THE DOWNLOAD OR INSTALL. IF YOU DO NOT ACCEPT THESE LICENSE TERMS, CLICK "I DO NOT ACCEPT," AND DO NOT DOWNLOAD OR INSTALL THIS SOFTWARE.

NON-EXCLUSIVE SOFTWARE LICENSE AGREEMENT

FOR MICROCHIP MEMORY DISK DRIVE FILE SYSTEM SOFTWARE VERSION 1.2

IMPORTANT - READ CAREFULLY.

This Nonexclusive Software License Agreement ("Agreement") is a contract between you, your heirs, successors and assigns ("Licensee") and Microchip Technology Incorporated, a Delaware corporation, with a principal place of business at 2355 W. Chandler Blvd., Chandler, AZ 85224-6199, and its subsidiary, Microchip Technology (Barbados) Incorporated (collectively, "Microchip") for Microchip's Memory Disk Drive File System Software version 1.2 ("Software") and accompanying

documentation ("Documentation").

The Software and Documentation are licensed under this Agreement and not sold. U.S. copyright laws and international copyright treaties, and other intellectual property laws and treaties protect the Software and Documentation. Microchip reserves all rights not expressly granted to Licensee in this Agreement.

1. Compliance with Third Party Requirements. Licensee acknowledges that it is Licensee's responsibility to obtain a copy of, familiarize itself fully with, and comply with the requirements and licensing obligations applicable to third party tools, systems and/or specifications including, but not limited to, flash-based media and FAT files systems available from Compact Flash Association, SD Card Association, Multi Media Card Association, and Microsoft Corporation. Microchip is not responsible and shall not be held responsible in any manner for Licensee's failure to comply with such requirements, including applicable license terms, if any.

2. License and Sublicense Grant.

(a) Definitions. As used in this Agreement, the following terms shall have the meanings defined below:

(i) "Microchip Products" means Microchip microcontrollers and digital signal controllers that use or implement the Software.

(ii) "Object Code" means the Software computer programming code provided by Microchip that is in binary form (including related documentation, if any), and error corrections, improvements and updates to such code provided by Microchip in its sole discretion, if any. (iii) "Source Code" means the Software computer programming code provided by Microchip that may be printed out or displayed in human readable form (including related programmer comments and documentation, if any), and error corrections, improvements, updates, modifications and derivatives of such code developed by Microchip, Licensee or Third Party (defined below).

(b) Software License Grant. Subject to the terms of this Agreement, Microchip grants strictly to Licensee a personal, worldwide, nonexclusive, non-transferable limited license to use, modify, copy and distribute the Software only when embedded in a Microchip Product that is either integrated into: (i) Licensee's product (hereafter "Licensee Product"); or (ii) Third Party Product pursuant to Section 2(d) below.

(c) Documentation License Grant. Subject to the terms of this Agreement, Microchip grants strictly to Licensee a personal, worldwide, non-exclusive, non-transferable limited license to use the Documentation in support of Licensee's authorized use of the Software.

(d) Sublicense Grants. Subject to the terms of this Agreement, Licensee may grant a limited sublicense to a third party ("Third Party") to use the Software as described below only IF such Third Party expressly agrees in writing to be bound by terms of confidentiality and limited use that are no broader in scope and duration than the confidentiality and limited use terms of this Agreement:

(i) Third Party may modify Source Code for Licensee.

(ii) Third Party may program Software into Microchip Products for Licensee.

(iii) Third Party may use the Software to develop and/or manufacture Licensee Products.

(iv) Third Party may use the Software to develop and/or manufacture Third Party Products where either: (x) the sublicensed Software contains Source Code modified or otherwise optimized by Licensee for integration into Third Party Products; and/or (y) the sublicensed Software is programmed into Microchip Products by Licensee on behalf of such Third Party.

(v) Third Party may use the Documentation in support of Third Party's authorized use of the Software in conformance with this Section 2(d).

As used in this Agreement, "Licensee Products" means Licensee products that use or incorporate Microchip Products. "Third Party Products" means Third party products that use or incorporate Microchip Products.

Licensee shall be responsible for any Third Party breach of the foregoing sublicense rights.

(e) Audit. Authorized representatives of Microchip shall have the right to reasonably inspect Licensee's premises and to audit Licensee's records and inventory of Licensee Products, whether located on Licensee's premises or elsewhere at any time, announced or unannounced, and in its sole and absolute discretion, in order to ensure Licensee's adherence to the terms of this Agreement.

3. Licensee Obligations.

(a) Licensee will ensure Third Party compliance with the terms of this Agreement.

(b) Licensee will not: (i) engage in unauthorized use, modification, disclosure or distribution of Software or Documentation, or its derivatives; (ii) use all or any portion of the Software, Documentation, or its derivatives except in conjunction with Microchip Products or Third Party Products; or (iii) reverse engineer (by disassembly, decompilation or otherwise) Software or any portion thereof.

(c) Licensee must include Microchip's copyright, trademark and other proprietary notices in all copies of the Software, Documentation, and its derivatives. Licensee may not remove or alter any Microchip copyright or other proprietary rights notice posted in any portion of the Software or Documentation.

(d) Licensee will defend, indemnify and hold Microchip and its subsidiaries harmless from and against any and all claims, costs, damages, expenses (including reasonable attorney's fees), liabilities, and losses, including without limitation product liability claims, directly or indirectly arising from or related to: (i) the use, modification, disclosure or distribution of the Software, Documentation or any intellectual property rights related thereto; (ii) the use, sale and distribution of Licensee Products or Third Party Products; and (iii) breach of Sections 1 or 2 of this Agreement. THE FOREGOING STATES THE SOLE AND EXCLUSIVE LIABILITY OF THE PARTIES FOR INTELLECTUAL PROPERTY INFRINGEMENT.

4. Confidentiality.

(a) Licensee agrees that the Software (including but not limited to the Source Code, Object Code and library files) and its derivatives, Documentation and underlying inventions, algorithms, know-how and ideas relating to the Software and the Documentation are proprietary information belonging to Microchip and its licensors ("Proprietary Information"). Except as expressly and unambiguously allowed herein, Licensee will hold in confidence and not use or disclose any Proprietary Information and shall similarly bind its employees and Third Party(ies) in writing. Proprietary Information shall not include information that: (i) is in or enters the public domain without breach of this Agreement and through no fault of the receiving party; (ii) the receiving party was legally in possession of prior to receiving it; (iii) the receiving party can demonstrate was developed by it independently and without use of or reference to the disclosing party's Proprietary Information; or (iv) the receiving party receives from a third party without restriction on disclosure. If Licensee is required to disclose Proprietary Information by law, court order, or government agency, such disclosure shall not be deemed a breach of this Agreement provided that Licensee gives Microchip prompt notice of such requirement in order to allow Microchip to object or limit such disclosure, Licensee cooperates with Microchip to protect Proprietary Information, and Licensee complies with any protective order in place and discloses only the information required by process of law.

(b) Licensee agrees that the provisions of this Agreement regarding unauthorized use and nondisclosure of the Software, Documentation and related Proprietary Rights are necessary to protect the legitimate business interests of Microchip and its licensors and that monetary damages alone cannot adequately compensate Microchip or its licensors if such provisions are violated. Licensee, therefore, agrees that if Microchip alleges that Licensee or Third Party has breached or violated such provision then Microchip will have the right to petition for injunctive relief, without the requirement for the posting of a bond, in addition to all other remedies at law or in equity.

5. Ownership of Proprietary Rights.

(a) Microchip and its licensors retain all right, title and interest in and to

the Software and Documentation ("Proprietary Rights") including, but not limited to: (i) patent, copyright, trade secret and other intellectual property rights in the Software, Documentation, and underlying technology; (ii) the Software as implemented in any device or system, all hardware and software implementations of the Software technology (expressly excluding Licensee and Third Party code developed and used in conformance with this Agreement solely to interface with the Software and Licensee Products and/or Third Party Products); and (iii) all copies and derivative works thereof (by whomever produced). Further, copies and derivative works shall be considered works made for hire with ownership vesting in Microchip on creation. To the extent such modifications and derivatives do not qualify as a "work for hire," Licensee hereby irrevocably transfers, assigns and conveys the exclusive copyright thereof to Microchip, free and clear of any and all liens, claims or other encumbrances, to the fullest extent permitted by law. Licensee and Third Party use of such modifications and derivatives is limited to the license rights described in Section 2 above.

(b) Licensee shall have no right to sell, assign or otherwise transfer all or any portion of the Software, Documentation or any related intellectual property rights except as expressly set forth in this Agreement.

6. Termination of Agreement. Without prejudice to any other rights, this Agreement terminates immediately, without notice by Microchip, upon a failure by License or Third Party to comply with any provision of this Agreement. Further, Microchip may also terminate this Agreement upon reasonable belief that Licensee or Third Party have failed to comply with this Agreement. Upon termination, Licensee and Third Party will immediately stop using the Software, Documentation, and derivatives thereof, and immediately destroy all such copies.

7. Warranties and Disclaimers. THE SOFTWARE AND

DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A PARTICULAR PURPOSE. MICROCHIP AND ITS LICENSORS ASSUME NO RESPONSIBILITY FOR THE ACCURACY, RELIABILITY OR APPLICATION OF THE SOFTWARE OR DOCUMENTATION. MICROCHIP AND ITS LICENSORS DO NOT WARRANT THAT THE SOFTWARE WILL MEET REQUIREMENTS OF LICENSEE OR THIRD PARTY, BE UNINTERRUPTED OR ERROR-FREE. MICROCHIP AND ITS LICENSORS HAVE NO OBLIGATION TO CORRECT ANY DEFECTS IN THE SOFTWARE. LICENSEE AND THIRD PARTY ASSUME THE ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE AND DOCUMENTATION PROVIDED UNDER THIS AGREEMENT.

8. Limited Liability. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL OR EQUITABLE THEORY FOR ANY DIRECT OR INDIRECT DAMAGES OR EXPENSES INCLUDING BUT NOT LIMITED TO INCIDENTAL, SPECIAL, INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS, TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS. The aggregate and cumulative liability of Microchip and its licensors for damages hereunder will in no event exceed \$1000 or the amount Licensee paid Microchip for the Software and Documentation, whichever is greater. Licensee acknowledges that the foregoing limitations are reasonable and an essential part of this Agreement.

9. General.

(a) Governing Law, Venue and Waiver of Trial by Jury. THIS AGREEMENT SHALL BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF THE STATE OF ARIZONA AND THE UNITED STATES WITHOUT REGARD TO CONFLICTS OF LAWS PROVISIONS. Licensee agrees that any disputes arising out of or related to this Agreement, Software or Documentation shall be brought in the courts of State of Arizona. The parties agree to waive their rights to a jury trial in actions relating to this Agreement.

(b) Attorneys' Fees. If either the Microchip or Licensee employs attorneys to enforce any rights arising out of or relating to this Agreement, the prevailing party shall be entitled to recover its reasonable attorneys' fees, costs and other expenses.

(c) Entire Agreement. This Agreement shall constitute the entire agreement between the parties with respect to the subject matter hereof. It shall not be modified except by a written agreement signed by an authorized representative of the Microchip.

(d) Severability. If any provision of this Agreement shall be held by a court of competent jurisdiction to be illegal, invalid or unenforceable, that provision shall be limited or eliminated to the minimum extent necessary so that this Agreement shall otherwise remain in full force and effect and enforceable.

(e) Waiver. No waiver of any breach of any provision of this Agreement shall constitute a waiver of any prior, concurrent or subsequent breach of the same or any other provisions hereof, and no waiver shall be effective unless made in writing and signed by an authorized representative of the waiving party. (f) Export Regulation. Licensee agrees to comply with all export laws and restrictions and regulations of the Department of Commerce or other United States or foreign agency or authority.

(g) Survival. The indemnities and obligations of confidentiality herein, and any right of action for breach of this Agreement prior to termination shall survive any termination of this Agreement.

(h) Assignment. Neither this Agreement nor any rights, licenses or obligations hereunder, may be assigned by Licensee without the prior written approval of Microchip except pursuant to a merger, sale of all assets of Licensee or other corporate reorganization, provided that assignee agrees in writing to be bound by the Agreement.

(i) Restricted Rights. Use, duplication or disclosure by the United States Government is subject to restrictions set forth in subparagraphs (a) through (d) of the Commercial Computer-Restricted Rights clause of FAR 52.227-19 when applicable, or in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, and in similar clauses in the NASA FAR Supplement. Contractor/manufacturer is Microchip Technology Inc., 2355 W. Chandler Blvd., Chandler, AZ 85225-6199.

If Licensee has any questions about this Agreement, please write to Microchip Technology Inc., 2355 W. Chandler Blvd., Chandler, AZ 85224-6199 USA, ATTN: Marketing.

License Rev. No. 01-081407

License Agreements > <u>MDD File System</u>

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USB Device Library Help

Contents | Index | Home

Previous

Graphics

License Agreement for usage of the Microchip Graphics Library v1.52

Remarks

IMPORTANT:

MICROCHIP IS WILLING TO LICENSE THIS GRAPHICS LIBRARY SOFTWARE

AND ACCOMPANYING DOCUMENTATION OFFERED TO YOU ONLY ON THE

CONDITION THAT YOU ACCEPT ALL OF THE FOLLOWING TERMS. TO

ACCEPT THE TERMS OF THIS LICENSE, CLICK "I ACCEPT" AND PROCEED

WITH THE DOWNLOAD OR INSTALL. IF YOU DO NOT ACCEPT THESE

LICENSE TERMS, CLICK "I DO NOT ACCEPT" AND DO NOT DOWNLOAD OR

INSTALL THIS SOFTWARE.

NON-EXCLUSIVE SOFTWARE LICENSE AGREEMENT FOR

MICROCHIP GRAPHICS LIBRARY SOFTWARE Version 1.52

IMPORTANT - READ CAREFULLY

This Nonexclusive Software License Agreement ("Agreement") is contract between you,

your heirs, agents, successors and assigns ("Licensee") and Microchip Technology

Incorporated, a Delaware corporation, with a principal place of business

at 2355 W.

Chandler Blvd., Chandler, AZ 85224-6199, and its subsidiary, Microchip Technology

(Barbados) Incorporated (collectively, "Microchip") for Microchip's Graphics Library

Software Version 1.52 ("Software") and accompanying documentation, including images

and any other graphic resources provided by Microchip ("Documentation").

The Software and Documentation are licensed under this Agreement and not sold. U.S.

copyright laws and international copyright treaties, and other intellectual property laws

and treaties protect the Software and Documentation. Microchip reserves all rights not

expressly granted to Licensee in this Agreement.

1. License and Sublicense Grant.

Definitions. As used this Agreement, the following terms shall have the meanings defined

below:

"Microchip Product" means Microchip microcontroller and digital signal controllers.

"Object Code" means the Software computer programming code provided by Microchip

that is in binary form (including related documentation, if any) and error corrections,

improvements and updates to such code provided by Microchip in its

sole discretion, if

any.

"Source Code" means the Software computer programming code provided by Microchip

that may be printed out or displayed in human readable form (including related

programmer comments and documentation, if any), and error corrections, improvements,

updates, modifications and derivatives of such code developed by Microchip, Licensee or

Third Party.

(b) Software License Grant. Subject to the terms of this Agreement, Microchip grants

strictly to Licensee a personal, worldwide, non-exclusive, non-transferable limited license

to use, modify, copy and distribute the Software only when embedded on a Microchip

Product that is integrated into Licensee's product (hereafter "Licensee Product") or Third

Party Product pursuant to Section 2(d) below.

Any portion of the Software (including derivatives or modifications thereof) may not be:

embedded on a non-Microchip microcontroller or digital signal controller;

distributed in Source Code or Object Code, except as described in Section 2(d) below.

(c) Documentation License Grant. Subject to all of the terms and

conditions of this

Agreement, Microchip grants strictly to Licensee a perpetual, worldwide, non-exclusive

license to use the Documentation within Licensee's company for purposes of developing

and supporting Conforming Systems.

(d) Sublicense Grants. Subject to terms of this Agreement, Licensee may grant a limited

sublicense to a third party ("Third Party") to use the Software as described below only IF

such Third Party expressly agrees to be bound by terms of confidentiality and limited use

that are no broader in scope and duration than the confidentiality and limited use terms of

this Agreement:

Third Party may modify Source Code for Licensee.

Third Party may program Software into Microchip Products for Licensee.

Third Party may use Software to develop and/or manufacture Licensee Product.

Third Party may use Software to develop and/or manufacture Third Party Products where

either: (x) the sublicensed Software contains Source Code modified or otherwise

optimized by Licensee for Third Party use; or (y) the sublicensed Software is

programmed into Microchip Products by Licensee on behalf of such

Third Party.

Third Party may use the Documentation in support of Third Party's authorized use of the

Software in conformance with this Section 2(d).

(e) Audit. Authorized representatives of Microchip shall have the right to reasonably

inspect Licensee's premises and to audit Licensee's records and inventory of Licensee

Products, whether located on Licensee's premises or elsewhere at any time, announced or

unannounced, and in its sole and absolute discretion, in order to ensure Licensee's

adherence to the terms of this Agreement.

2. Licensee's Obligations.

Licensee will ensure Third Party compliance with the terms of this Agreement.

Licensee will not: (i) engage in unauthorized use, modification, disclosure or distribution

of Software or Documentation, or its derivatives; (ii) use all or any portion of the

Software, Documentation, or its derivatives except in conjunction with Microchip

Products; or (iii) reverse engineer (by disassembly, decompilation or otherwise) Software

or any portion thereof.

(c) Licensee must include Microchip's copyright, trademark and other proprietary notices

in all copies of the Software, Documentation, and its derivatives. Licensee may not

remove or alter any Microchip copyright or other proprietary rights notice posted in any

portion of the Software or Documentation.

(d) Licensee will defend, indemnify and hold Microchip and its subsidiaries harmless

from and against any and all claims, costs, damages, expenses (including reasonable

attorney's fees), liabilities, and losses, including without limitation product liability

claims, directly or indirectly arising from or related to: (i) the use, modification,

disclosure or distribution of the Software, Documentation or any intellectual property

rights related thereto; and (ii) the sale of Licensee Products or Third Party Products. THE

FOREGOING STATES THE SOLE AND EXCLUSIVE LIABILITY OF THE PARTIES

FOR INTELLECTUAL PROPERTY RIGHTS INFRINGEMENT.

3. Confidentiality.

(a) Licensee agrees that the Software (including but not limited to the Source Code,

Object Code and library files) and its derivatives, Documentation and underlying

inventions, algorithms, know-how and ideas relating to the Software and the

Documentation are proprietary information belonging to Microchip and its licensors

("Proprietary Information"). Except as expressly and unambiguously allowed herein,

Licensee will hold in confidence and not use or disclose any Proprietary Information and

shall similarly bind its employees and Third Party(ies) in writing. Proprietary Information

shall not include information that: (i) is in or enters the public domain without breach of

this Agreement and through no fault of the receiving party; (ii) the receiving party was

legally in possession of prior to receiving it; (iii) the receiving party can demonstrate was

developed by it independently and without use of or reference to the disclosing party's

Proprietary Information; or (iv) the receiving party receives from a third party without

restriction on disclosure. If Licensee is required to disclose Proprietary Information by

law, court order, or government agency, such disclosure shall not be deemed a breach of

this Agreement provided that Licensee gives Microchip prompt notice of such

requirement in order to allow Microchip to object or limit such disclosure, Licensee

cooperates with Microchip to protect Proprietary Information, and Licensee complies

with any protective order in place and discloses only the information required by process

of law.

(b) Licensee agrees that the provisions of this Agreement regarding unauthorized use and

nondisclosure of the Software, Documentation and related Proprietary Rights are

necessary to protect the legitimate business interests of Microchip and its licensors and

that monetary damages alone cannot adequately compensate Microchip or its licensors if

such provisions are violated. Licensee, therefore, agrees that if Microchip alleges that

Licensee or Third Party has breached or violated such provision then Microchip will have

the right to petition for injunctive relief, without the requirement for the posting of a

bond, in addition to all other remedies at law or in equity.

4. Ownership of Proprietary Rights.

(a) Microchip and its licensors retain all right, title and interest in and to the Software and

Documentation ("Proprietary Rights") including, but not limited to: (i) patent, copyright,

trade secret and other intellectual property rights in the Software, Documentation, and

underlying technology; (ii) the Software as implemented in any device or system, all

hardware and software implementations of the Software technology (expressly excluding

Licensee and Third Party code developed and used in conformance with this Agreement

solely to interface with the Software and Licensee Products and/or Third Party Products);

and (iii) all copies and derivative works thereof (by whomever produced). Further, copies

and derivative works shall be considered works made for hire with ownership vesting in

Microchip on creation. To the extent such modifications and derivatives do not qualify as

a "work for hire," Licensee hereby irrevocably transfers, assigns and conveys the

exclusive copyright thereof to Microchip, free and clear of any and all liens, claims or

other encumbrances, to the fullest extent permitted by law. Licensee and Third Party use

of such modifications and derivatives is limited to the license rights described in Section

1 above.

(b) Licensee shall have no right to sell, assign or otherwise transfer all or any portion of

the Software, Documentation or any related intellectual property rights except as

expressly set forth in this Agreement.

5. Termination of Agreement. Without prejudice to any other rights, this Agreement

terminates immediately, without notice by Microchip, upon a failure by License or Third

Party to comply with any provision of this Agreement. Further, Microchip may also

terminate this Agreement upon reasonable belief that Licensee or Third Party have failed

to comply with this Agreement. Upon termination, Licensee and Third Party will

immediately stop using the Software, Documentation, and derivatives thereof, and

immediately destroy all such copies.

6. Warranties and Disclaimers. THE SOFTWARE AND DOCUMENTATION ARE

PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS

OR IMPLIED, INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF

MERCHANTABILITY, TITLE, NON-INFRINGEMENT AND FITNESS FOR A

PARTICULAR PURPOSE. MICROCHIP AND ITS LICENSORS ASSUME NO

RESPONSIBILITY FOR THE ACCURACY, RELIABILITY OR APPLICATION OF

THE SOFTWARE OR DOCUMENTATION. MICROCHIP AND ITS LICENSORS DO

NOT WARRANT THAT THE SOFTWARE WILL MEET REQUIREMENTS OF

LICENSEE OR THIRD PARTY, BE UNINTERRUPTED OR ERROR-

FREE.

MICROCHIP AND ITS LICENSORS HAVE NO OBLIGATION TO CORRECT ANY

DEFECTS IN THE SOFTWARE. LICENSEE AND THIRD PARTY ASSUME THE

ENTIRE RISK ARISING OUT OF USE OR PERFORMANCE OF THE SOFTWARE

AND DOCUMENTATION PROVIDED UNDER THIS AGREEMENT.

7. Limited Liability. IN NO EVENT SHALL MICROCHIP OR ITS LICENSORS BE

LIABLE OR OBLIGATED UNDER CONTRACT, NEGLIGENCE, STRICT

LIABILITY, CONTRIBUTION, BREACH OF WARRANTY, OR OTHER LEGAL OR

EQUITABLE THEORY FOR ANY DIRECT OR INDIRECT DAMAGES OR

EXPENSES INCLUDING BUT NOT LIMITED TO INCIDENTAL, SPECIAL,

INDIRECT, PUNITIVE OR CONSEQUENTIAL DAMAGES, LOST PROFITS OR

LOST DATA, COST OF PROCUREMENT OF SUBSTITUTE GOODS,

TECHNOLOGY, SERVICES, OR ANY CLAIMS BY THIRD PARTIES (INCLUDING

BUT NOT LIMITED TO ANY DEFENSE THEREOF), OR OTHER SIMILAR COSTS.

The aggregate and cumulative liability of Microchip and its licensors for damages

hereunder will in no event exceed \$1000 or the amount Licensee paid Microchip for the

Software and Documentation, whichever is greater. Licensee acknowledges that the

foregoing limitations are reasonable and an essential part of this Agreement.

8. Third Party Technology.

(a) This software is based in part on the work of the Independent JPEG Group. If such

software is provided in source code format, then the applicable license terms are posted in

the 'IJG License.pdf' file which can be found in: <Installed Folder>\Microchip\Image

Decoders.

(b) With respect to the uncompressed GIFs, the Graphics Interchange ${\sf Format}{\mathbb O}$ is

copyright of CompuServe Incorporated and the GIF(SM) is a service mark of

CompuServe Incorporated.

9. General.

(a) Governing Law, Venue and Waiver of Trial by Jury. THIS AGREEMENT SHALL

BE GOVERNED BY AND CONSTRUED UNDER THE LAWS OF THE STATE OF

ARIZONA AND THE UNITED STATES WITHOUT REGARD TO CONFLICTS OF

LAWS PROVISIONS. Licensee agrees that any disputes arising out of

or related to this

Agreement, Software or Documentation shall be brought in the courts of State of

Arizona. The parties agree to waive their rights to a jury trial in actions relating to this

Agreement.

(b) Attorneys' Fees. If either the Microchip or Licensee employs attorneys to enforce any

rights arising out of or relating to this Agreement, the prevailing party shall be entitled to

recover its reasonable attorneys' fees, costs and other expenses.

(c) Entire Agreement. This Agreement shall constitute the entire agreement between the

parties with respect to the subject matter hereof. It shall not be modified except by a

written agreement signed by an authorized representative of the Microchip.

(d) Severability. If any provision of this Agreement shall be held by a court of competent

jurisdiction to be illegal, invalid or unenforceable, that provision shall be limited or

eliminated to the minimum extent necessary so that this Agreement shall otherwise

remain in full force and effect and enforceable.

(e) Waiver. No waiver of any breach of any provision of this Agreement shall constitute a

waiver of any prior, concurrent or subsequent breach of the same or

any other provisions

hereof, and no waiver shall be effective unless made in writing and signed by an

authorized representative of the waiving party

(f) Export Regulation. Licensee agrees to comply with all export laws and restrictions and

regulations of the Department of Commerce or other United States or foreign agency or

authority.

(g) Survival. The indemnities and obligations of confidentiality herein, and any right of

action for breach of this Agreement prior to termination shall survive any termination of

this Agreement.

(h) Assignment. Neither this Agreement nor any rights, licenses or obligations hereunder,

may be assigned by Licensee without the prior written approval of Microchip except

pursuant to a merger, sale of all assets of Licensee or other corporate reorganization,

provided that assignee agrees in writing to be bound by the Agreement.

(i) Restricted Rights. Use, duplication or disclosure by the United States Government is

subject to restrictions set forth in subparagraphs (a) through (d) of the Commercial

Computer-Restricted Rights clause of FAR 52.227-19 when applicable,

or in

subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at

DFARS 252.227-7013, and in similar clauses in the NASA FAR Supplement.

Contractor/manufacturer is Microchip Technology Inc., 2355 W. Chandler Blvd.,

Chandler, AZ 85225-6199.

If Licensee has any questions about this Agreement, please write to Microchip

Technology Inc., 2355 W. Chandler Blvd., Chandler, AZ 85224-6199 USA, ATTN:

Marketing.

Microchip Graphics Library Software Version 1.52 Copyright $\ensuremath{\mathbb{C}}$ 2008 Microchip

Technology Inc. All rights reserved.

License Rev. (290808)

License Agreements > Graphics

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USB Device Library Help

Contents | Index | Home

Contents

Introduction MCHPFSUSB Device Library

Function Drivers

Communication Device Class (CDC)

Public API Members

Functions and Macros

CDCInitEP Function CDCTxService Function getsUSBUSART Function putrsUSBUSART Function putUSBUSART Function USBUSART Function USBUSARTISTxTrfReady Macro CDCSetLineCoding Macro CDCSetBaudRate Macro CDCSetCharacterFormat Macro CDCSetParity Macro

CDCSetDataSize Macro

Definitions, Constants, and Enums

NUM_STOP_BITS_1 Macro NUM_STOP_BITS_1_5 Macro NUM_STOP_BITS_2 Macro PARITY_EVEN Macro PARITY_MARK Macro PARITY_NONE Macro PARITY_ODD Macro PARITY_SPACE Macro

Variables

Depricated API Members

mUSBUSARTIsTxTrfReady Macro

Files

usb_function_cdc.h usb_function_cdc.c usb_config.h HardwareProfile.h

Human Interface Device (HID)

Public API Members

Functions and Macros

USBCheckHIDRequest Function HIDTxPacket Macro HIDTxHandleBusy Macro **HIDRxPacket Macro** HIDRxHandleBusy Macro Definitions, Constants, and Enums Variables Files usb_function_hid.c usb function hid.h usb config.h Mass Storage Device (MSD) **Public API Members Functions and Macros MSDTasks Function USBMSDInit Function** USBCheckMSDRequest Function **Definitions, Constants, and Enums** LUN FUNCTIONS Structure MSD WAIT Macro MSD DATA IN Macro MSD_DATA_OUT Macro MSD SEND CSW Macro Variables Files usb function msd.c usb function msd.h usb config.h **Vendor Class Public API Members Functions and Macros USBGenRead Macro USBGenWrite Macro** Definitions, Constants, and Enums Variables Files usb_function_generic.c usb_function_generic.h usb_config.h Stack

Public API Members

Functions and Macros

USBDeviceInit Function USBDeviceTasks Function USBGetDeviceState Macro USBGetRemoteWakeupStatus Macro USBIsDeviceSuspended Macro **USBEnableEndpoint Function USBHandleBusy Macro** USBHandleGetAddr Macro USBHandleGetLength Macro **USBCBInitEP** Function **USBCBSuspend Function** USBCBWakeFromSuspend Function USBCBCheckOtherReg Function **USBCBSendResume Function USBCBErrorHandler Function** USBCBStdSetDscHandler Function USBCB_SOF_Handler Function **USBCBEP0DataReceived Function**

Definitions, Constants, and Enums

DETACHED_STATE Macro ATTACHED_STATE Macro POWERED_STATE Macro DEFAULT_STATE Macro ADR_PENDING_STATE Macro ADDRESS_STATE Macro CONFIGURED_STATE Macro _DEFAULT Macro

_RWU Macro

_SELF Macro

Variables and Types

USB_HANDLE Macro

Files

usb_device.c usb.h usb_ch9.h usb_common.h usb_device.h usb_hal.h usb_hal_pic18.h usb_hal_pic24.h usb_hal_pic32.h usb_config.h HardwareProfile.h *License Agreements* MCHPFSUSB MDD File System Graphics

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright C 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USB Device Library Help

Contents | Index | Home

Index

 $_|A|C|D|E|G|H|!|L|M|N|P|S|U|V$

<u>_DEFAULT macro</u> <u>_RWU macro</u> _<u>SELF macro</u>

A

ADDRESS_STATE macro ADR_PENDING_STATE macro ATTACHED_STATE macro

С

CDCInitEP function CDCSetBaudRate macro CDCSetCharacterFormat macro CDCSetDataSize macro CDCSetLineCoding macro CDCSetParity macro CDCTxService function Communication Device Class (CDC) CONFIGURED_STATE macro

D

DEFAULT_STATE macro Definitions, Constants, and Enums Depricated API Members DETACHED_STATE macro Public API Members putrsUSBUSART function putsUSBUSART function putUSBUSART function

S

<u>Stack</u>

U

usb.h usb_ch9.h usb_common.h usb_config.h usb_device.c usb_device.h usb_function_cdc.c usb_function_cdc.h usb_function_generic.c usb_function_generic.h usb_function_hid.c

F

Files Function Drivers Functions and Macros

G

getsUSBUSART function Graphics

Η

HardwareProfile.h HIDRxHandleBusy macro HIDRxPacket macro HIDTxHandleBusy macro HIDTxPacket macro Human Interface Device (HID)

Introduction

L

LUN_FUNCTIONS structure

Μ

Mass Storage Device (MSD) MAX_LUN MCHPFSUSB MDD File System MSD_DATA_IN macro MSD_DATA_OUT macro MSD_SEND_CSW macro MSD_WAIT macro usb function hid.h usb function msd.c usb function msd.h usb hal.h usb hal pic18.h usb hal pic24.h usb hal pic32.h **USB HANDLE macro USBCB SOF Handler function** USBCBCheckOtherReg function **USBCBEP0DataReceived** function USBCBErrorHandler function **USBCBInitEP** function **USBCBSendResume function USBCBStdSetDscHandler** function **USBCBSuspend function USBCBWakeFromSuspend** function **USBCheckHIDRequest function USBCheckMSDRequest function USBDeviceInit function USBDeviceTasks function USBEnableEndpoint function USBGenRead** macro **USBGenWrite macro** USBGetDeviceState macro <u>USBGetRemoteWakeupStatus</u> macro **USBHandleBusy macro** USBHandleGetAddr macro <u>USBHandleGetLength macro</u> **USBIsDeviceSuspended macro USBMSDInit function**

MSDTasks function mUSBUSARTIsTxTrfReady macro

Ν

NUM_STOP_BITS_1 macro NUM_STOP_BITS_1_5 macro NUM_STOP_BITS_2 macro

Ρ

PARITY_EVEN macro PARITY_MARK macro PARITY_NONE macro PARITY_ODD macro PARITY_SPACE macro POWERED_STATE macro

Microchip MCHPFSUSB v2.3 - Sept 20, 2008 Copyright © 2008 Microchip Technology, Inc. All rights reserved.

Contents | Index | Home

USBUSARTIsTxTrfReady macro

V

Variables Variables and Types Vendor Class