chipKIT™ USBDevice Class

Introduction

The chipKIT™ USBDevice class is fundamentally a direct mapping of the underlying Microchip USB Device Interface Routines. Buy understanding the Microchip USB Device Stack, the usage of the chipKIT™ USBDevice class will become clear. The chipKIT™ USBDevice class only implements a few additional class constructors, an initialization routine, and access to the event default Generic USB callback routine. Above and beyond that, all of the other methods are direct thunks to the underlying Microchip USB Device Interface Routines. The chipKIT™ USBDevice class methods will have the same base API names and parameter arguments as the underlying Microchip USB Device Interface Routines.

For more information on how to use the Microchip USB Device Stack, refer to http://ww1.microchip.com/downloads/en/DeviceDoc/MCHPFSUSB_FW_UG_51679a.pdf and the "*MCHPFSUSB Library Help.chm*" help files. For the documentation of the common USBDevice class methods (thunks), refer to the "*Library Interface (API) > Device/Peripheral > Device Stack > Interface Routines*" section in the "*MCHPFSUSB Library Help.chm*". A table of the mappings between the USBDevice class and the underlying Microchip USB Device Interface Routines can be found at the end of this document.

To understand the chipKIT™ USBDevice class constructors and initialization routine, it is necessary to understand that whenever a USB Device connects to a USB host the USB host starts to request information from the USB device to describe itself and usage. This is done by a series of events that the host request from the device. Some of this initial step-up is common across all devices, for example the initial passing of the device's descriptor tables and the setup of the control endpoint 0. But beyond that, the communication becomes more specific for each device type. Since the host is driving the communication, the device is queried to respond to the host, and as a result, the USB Stack has implemented the processing of handling these events in a callback routine that must be supplied by the application. A default callback routine has been provided that may in many cases be able to handle the initial communication of the device with the host up to the point where the device becomes configured; which is after the time-critical passing of the device descriptors and the setting up of control endpoint 0. In many cases, depending on the device, the remaining device specific configuration can be completed in a non-time critical communication after endpoint 0 is configured. The default callback routine can only stand on its own for the Generic USB Device; this is an uncommon device as Microsoft Windows® does not have a default Generic USB Device Driver preinstalled on Windows. However, the Generic USB device is the simplest of all USB devices and you can get a Microsoft Windows® Generic USB Device Driver from the Windows SDK. As a convenience, the WinUSB device driver as provided by the Microchip SimpleWinUSB example is provided along with the Generic USB Device example sketch. You will need to install this WinUSB device driver on your computer before the Generic USB Device example sketch will run. However, Windows does preinstalled a HID USB Device Driver. This example sketch is a little more complicated than the Generic USB Device, but you can still use the default callback routine to configure endpoint 0 for you. However, you must add additional callback event handling to handle the HID portion of the configuration. Two examples have been provided, a Generic USB Device, and a Custom HID Device. It might be easier to start with the Custom HID Device example as this will not require you to install any Device Drivers in Windows. The example also shows how you can leverage the default callback routine in your application provided callback routine to help you through the time-critical configuration.

The chipKIT™ USBDevice class constructor can be constructed to use only the default callback routine (which would only work for a Generic USB Device), or to specify your own application specific callback routine.

Once the chipKIT™ USB Device class is constructed, the USB Stack needs to be initialized. Initialization enables hardware specific configuration, resets the state of the stack, and starts listening for host events. The chipKIT™ USB Device class optionally provides the ability to block until the USB Device is configured with the host. This may be desirable as this dedicates all of the chipKIT™ processor resources to the host request events during this time-critical period while being configured. Remember, that during this time, your application specified callback routine (if any) will be called; so you will want to write the callback routine to respond quickly.

The chipKIT™ initialization routine InitializeSystem() will automatically call DeviceInit(). Typically your sketch should use InitializeSystem() and not DeviceInit(), however it is provided should you want to initialize the USB Stack manually. InitializeSystem() and DeviceInit() are mutually exclusive, do not call both of them.

USBDevice Constructors

USBDevice usb;
USBDevice usb(MyCallBackRoutine);

Parameters:

> **MyCallBackRoutine** - This is the application supplied callback routine to handle host request events. Optional: if not specified the default callback routine will be used.

This constructs the USB Device Class. Only one USBDevice class can be instantiated per Sketch.

USBDevice Methods

void InitializeSystem(boolean fWaitUntilConfigured)

Parameters:

> **fWaitUntilConfigured** - If true will block until the USB stack is in the “configured” state. Typically this is after control endpoint 0 is setup. The application specified callback routine as specified in the constructor (if any) will be called during configuration. If fWaitUntilConfigured is set to false, InitializeSystem() will return immediately after starting to listen for host request events, but before the USB connection with the host is configured.

Returns:

> Nothing.

Initializes the hardware, resets the USB Stack, and starts listening for host request events. This will automatically call DeviceInit(). DeviceInit() and InitializeSystem() are mutually exclusive; typically InitializeSystem() is used unless manual initialization of the USB stack is required.

*boolean DefaultCBEventHandler(USB_EVENT event, void *pdata, word size)*

Parameters:

> **event** - The type of event as defined by USB_EVENT.
> **pdata** - A pointer to the data being passed; event specific.
> **size** - The length of the data; event specific.

Returns:

> Always true. When writing your own callback routine the return value is user defined.

Calls the Generic USB default callback routine. This may allow the application writer to leverage the default callback routine through configuration with the host. If used, typically the application callback routine would call the default callback routine first and then check the event to see if it is an event that the application must handle. See the Microchip USB_APPLICATION_EVENT_HANDLER for more information. This is a complete callback handler for a USB Generic Device.

ChipKIT™ USBDevice Class Method Mappings to the Underlying Microchip USB Device Interface Routines

Refer to the "*Library Interface (API) > Device/Peripheral > Device Stack > Interface Routines*" section in the "*MCHPFSUSB Library Help.chm*" for documentation of these methods.

chipKIT™ USB Device Class Method	Microchip USB Device Interface Routine
void DeviceTasks(void)	void USBDeviceTasks(void)
void DeviceInit(void)	void USBDeviceInit(void)
boolean GetSuspendState(void)	BOOL USBGetSuspendState(void)
void EnableEndpoint(byte ep, byte options)	void USBEnableEndpoint(BYTE ep, BYTE options)
USB_HANDLE TransferOnePacket(byte ep, byte dir, byte* data, byte len)	USB_HANDLE USBTransferOnePacket(BYTE ep, BYTE dir,BYTE* data,BYTE len)
void StallEndpoint(byte ep, byte dir)	void USBStallEndpoint(BYTE ep, BYTE dir)
void CancelIO(byte endpoint)	void USBCancelIO(BYTE endpoint)
boolean GetRemoteWakeupStatus(void)	BOOL USBGetRemoteWakeupStatus(void)
USB_DEVICE_STATE GetDeviceState(void)	USB_DEVICE_STATE USBGetDeviceState(void)
boolean IsDeviceSuspended(void)	BOOL USBIsDeviceSuspended(void)
void SoftDetach(void)	void USBSoftDetach(void)
boolean HandleBusy(USB_HANDLE handle)	BOOL USBHandleBusy(USB_HANDLE handle)
word HandleGetLength(USB_HANDLE handle)	WORD USBHandleGetLength(USB_HANDLE handle)
void * HandleGetAddr(USB_HANDLE)	WORD USBHandleGetAddr(USB_HANDLE)
void EP0Transmit(byte options)	void USBEP0Transmit(BYTE options)
void EP0SendRAMPtr(byte* src, word size, byte Options)	void USBEP0SendRAMPtr(BYTE* src, WORD size, BYTE Options)
void EP0SendROMPtr(byte* src, word size, byte Options)	void USBEP0SendROMPtr(BYTE* src, WORD size, BYTE Options)
void EP0Receive(byte* dest, word size, void (*function))	void USBEP0Receive(BYTE* dest, WORD size, void (*function))
USB_HANDLE TxOnePacket(byte ep, byte* data, word len);	USB_HANDLE USBTxOnePacket(BYTE ep, BYTE* data, WORD len)
USB_HANDLE GenWrite(byte ep, byte* data, word len)	Same as TxOnePacket
USB_HANDLE RxOnePacket(byte ep, byte* data, word len)	USB_HANDLE USBRxOnePacket(BYTE ep, BYTE* data, WORD len)
USB_HANDLE GenRead(byte ep, byte* data, word len)	Same as RxOnePacket
void DeviceDetach(void)	void USBDeviceDetach(void)
void DeviceAttach(void)	void USBDeviceAttach(void)