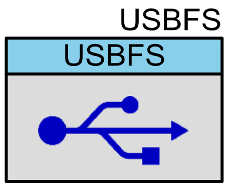
Full Speed USB (USBFS)

1.50

Features



- USB Full Speed device interface driver
- Support for interrupt, control, bulk, and isochronous transfer types
- Runtime support for descriptor set selection
- Optional USB string descriptors
- Optional USB HID class support
- Optional Boot Loader support

General Description

The USBFS component provides a USB full speed Chapter 9 compliant device framework. The component provides a low level driver for the control endpoint that decodes and dispatches requests from the USB host. Additionally, this component provides a USBFS customizer to enable easy descriptor construction.

You have the option of constructing an HID based device or a generic USB Device. Select HID (and switch between HID and generic) by setting the Configuration/Interface descriptors.

Refer to the USB-IF device class documentation for additional information on descriptors (http://www.usb.org/developers/devclass/).

Note Cypress offers a set of USB development tools, called SuiteUSB, available free of charge when used with Cypress silicon. You can obtain SuiteUSB from the Cypress web site: http://www.cypress.com.

When to use a USBFS

Use the USBFS component when you want to provide your application with a USB 2.0 compliant device interface.

Quick Start

1. Drag a USBFS component from the Component Catalog onto your design.
2. Notice the clock errors in the Notice List window; double-click on an error to open the System Clock Editor.

PRELIMINARY

3. Configure the following clocks:
   - **ILO**: Select 100 kHz.
   - **IMO**: Select Osc 24.000 MHz.
   - **USB**: Enable and select IMOx2 – 48.000 MHz.

   Note If the selected device is PSoC 3 ES2 or PSoC 5, you must also configure the PLL to "Desired 33 MHz" and the Master Clock to "PLL_OUT (33.000 MHz)".
4. Select **Build** to generate APIs; refer to the Sample Firmware Source Code section for an example demonstrating the basic functionality of the USBFS component, as well as the basic set-up instructions.

Input/Output Connections

This section describes the various input and output connections for the USBFS. An asterisk (*) in the list of I/Os indicates that the I/O may be hidden on the symbol under the conditions listed in the description of that I/O.

sof – Output *

The Start-of-Frame (sof) output allows endpoints to identify the start of the frame and synchronize internal endpoint clocks to the host. This output is visible if the "out_sof" parameter in advanced configuration tab is set to enable.

Component Parameters

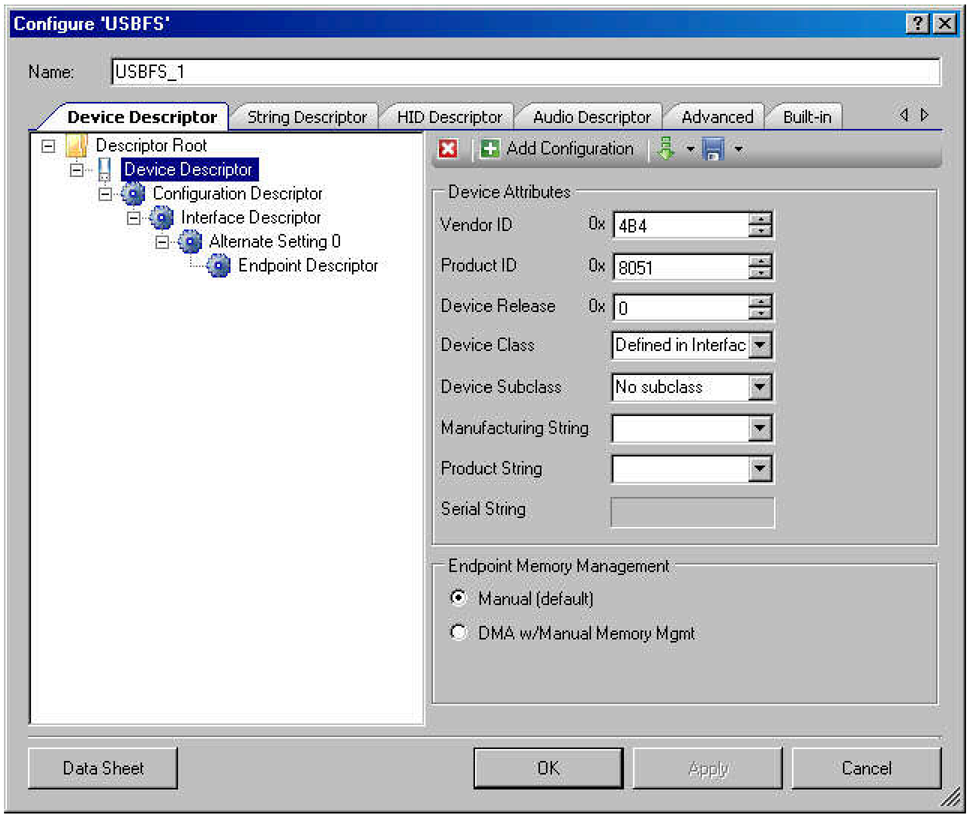
Drag a USBFS component onto your design and double-click it to open the Configure USBFS dialog.

The component is driven by information generated by the USBFS Configure dialog. This dialog – or customizer – facilitates the construction of the USB descriptors and integrates the information generated into the driver firmware used for device enumeration.

The USBFS component does not function without first running the wizard and selecting the appropriate attributes to describe your device. The code generator takes your device information and generates all of the needed USB Descriptors.

The Configure USBFS dialog contains the following tabs and settings:

Device Descriptor Tab


Configure 'USBFS'

Name: USBFS_1

Device Attributes

Vendor ID 0x 4B4

Product ID 0x 8051

Device Release 0x 0

Device Class: Defined in Interfac

Device Subclass: No subclass

Manufacturing String:

Product String:

Serial String:

Endpoint Memory Management

[✓ Manual (default)]

DMA w/Manual Memory Mgmt

Device Attributes

- Vendor ID – Your Company USB Vendor ID (obtained from USB-IF)

  Note Vendor ID 0x4B4 is a Cypress only VID and may be used for development purposes only. Products cannot be released using this VID; you must obtain your own VID.
- Product ID – Your Specific Product ID
- Device Release – Your Specific Device Release (Device ID)
- Device Class – Device Class is defined in Interface Descriptor or it is Vendor Specific
- Device Subclass – Dependent upon Device Class
- Manufacturing String – Manufacturer Specific Description String to be displayed when the device is attached.
- Product String – Product specific Description String to be displayed when the device is attached.
- Serial String

Endpoint Memory Management

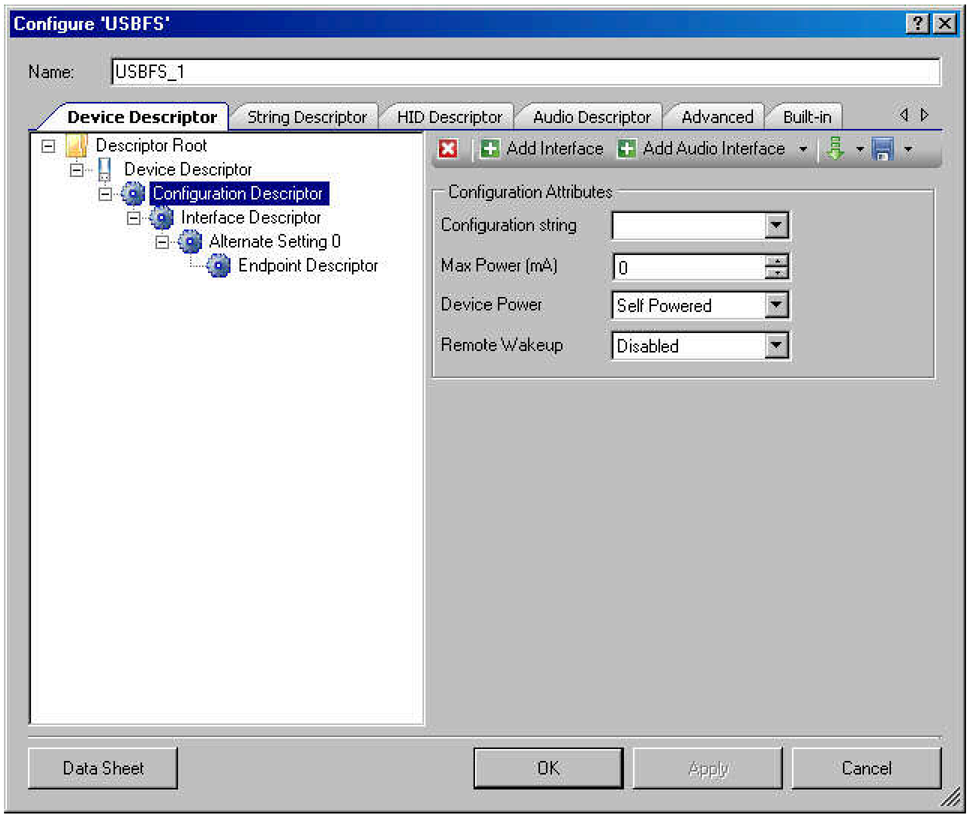
Some applications can benefit from using Direct Memory Access (DMA) to move data into and out of the endpoint memory buffers.

- Manual (default) – Select this option to use LoadInEP/ReadOutEP to load and unload the endpoint buffers.
- DMA w/Manual Memory Management – Select this option to expose the DMA interface registers, in addition to the LoadInEP/ReadOutEP functions.

  PSoC 3 does not support DMA transactions directly between USB endpoints and other peripherals. All DMA transactions involving USB endpoints (IN & OUT) must terminate or originate with main system memory.

  Applications requiring DMA directly between USB endpoints and other peripherals must use two DMA transactions. The two transactions move data to main system memory as an intermediate step between the USB endpoint and the other peripheral.

Configuration Descriptor



Name: USBFS_1

Configuration string:
Max Power (mA): 0
Device Power: Self Powered
Remote Wakeup: Disabled

Configuration Attributes

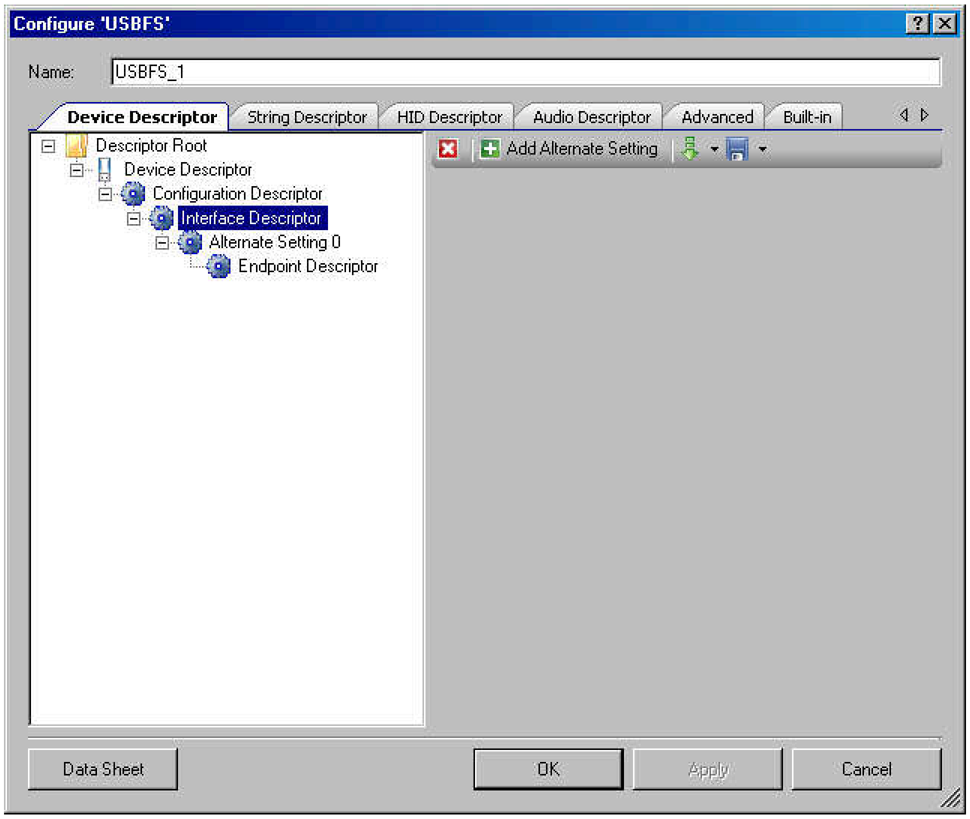
- Configuration string

- Max Power (mA) Enter the maximum power consumption of the USB device from the bus in this specific configuration when the device is fully operational.

  Note A device configuration reports whether the configuration is bus-powered or self-powered. Device status reports whether the device is currently self-powered. If a device is disconnected from its external power source, it updates device status to indicate that it is no longer self-powered. A device may not increase its power draw from the bus, when it loses its external power source, beyond the amount reported by its configuration.
- Device Power – Bus Powered or Self Powered Device. The USBFS does not support both usages simultaneously.
- Remote Wakeup – Enabled or Disabled

Interface Descriptor

This level is used to add and delete Interface Alternate Settings. The interfaces are configured in the Alternate Setting.

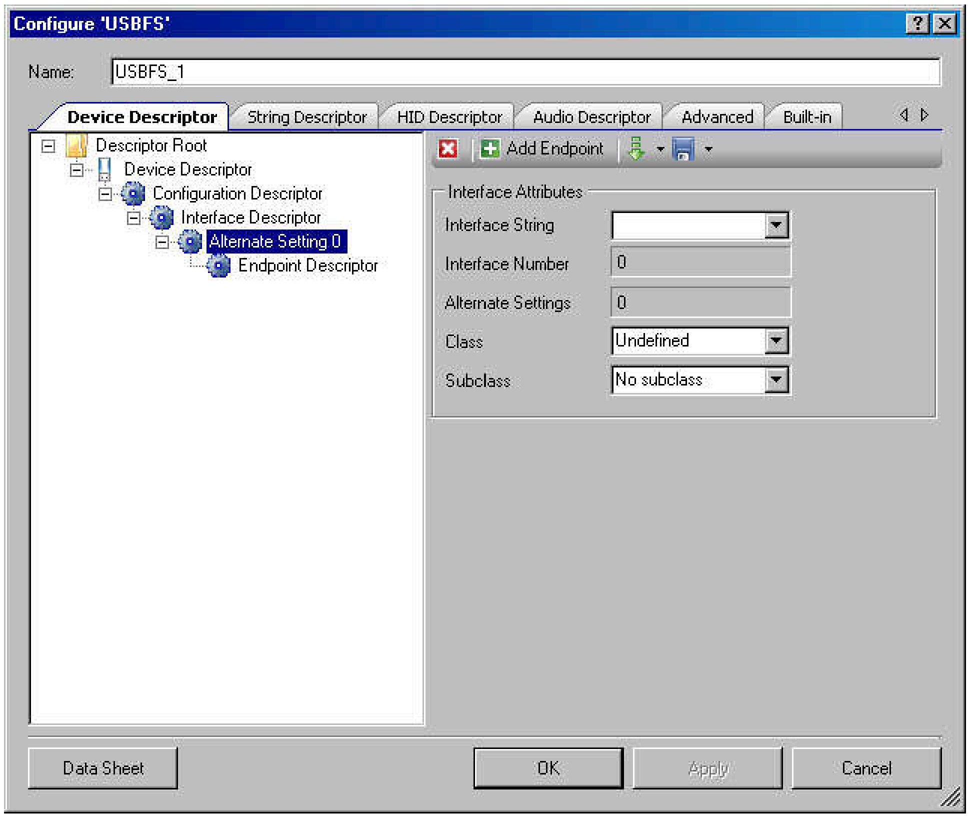


Alternate Setting 0 is automatically provided to configure your device. If your device will use isochronous endpoints, note that the USB 2.0 specification requires that all device default interface settings must not include any isochronous endpoints with non-zero data payload sizes. This is specified via Max Packet Size in the endpoint descriptor.

For isochronous devices, you should use an alternate interface setting other than the default Alternate Setting 0 to specify non-zero data payload sizes for isochronous endpoints.

Additionally, if your isochronous endpoints have a large data payload size, it is recommended that additional alternate configurations or interface settings be used to specify a range of data payload sizes. This increases the chance that the device can be used successfully in combination with other USB devices.

Interface Descriptor—Alternate Settings



Name: USBFS_1

Interface String	
Interface Number	0
Alternate Settings	0
Class	Undefined
Subclass	No subclass

Interface Attributes

- Interface String
- Interface Number — The interface number is computed by the customizer
- Alternate Settings — The alternate setting is computed by the customizer
- Class – HID, Vendor Specific or Undefined
- Subclass – Dependent upon the selected class

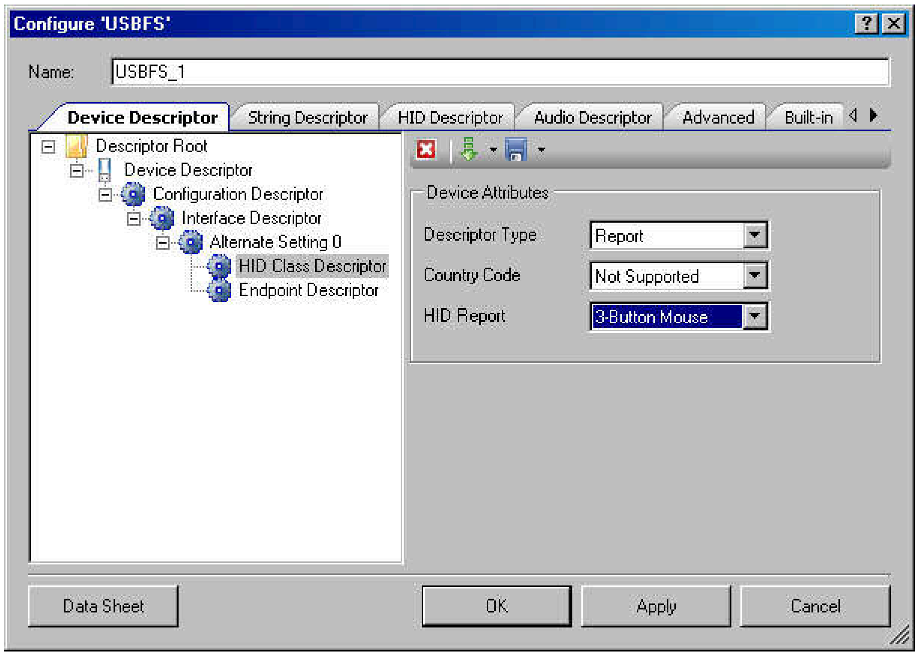
Note String Descriptors are optional. If a device does not support string descriptors, all references to string descriptors within the device, configuration and interface descriptors must be set to zero.

HID Class Descriptor

The HID Class Descriptor item does not display by default. It is used to add an HID Report to the Alternate Setting.

To Add HID Class Descriptor

1. Select an Alternate Setting item in the Descriptor Root tree.
2. Under **Interface Attributes** on the right, select "HID" for the **Class** field.



Name: USBFS_1

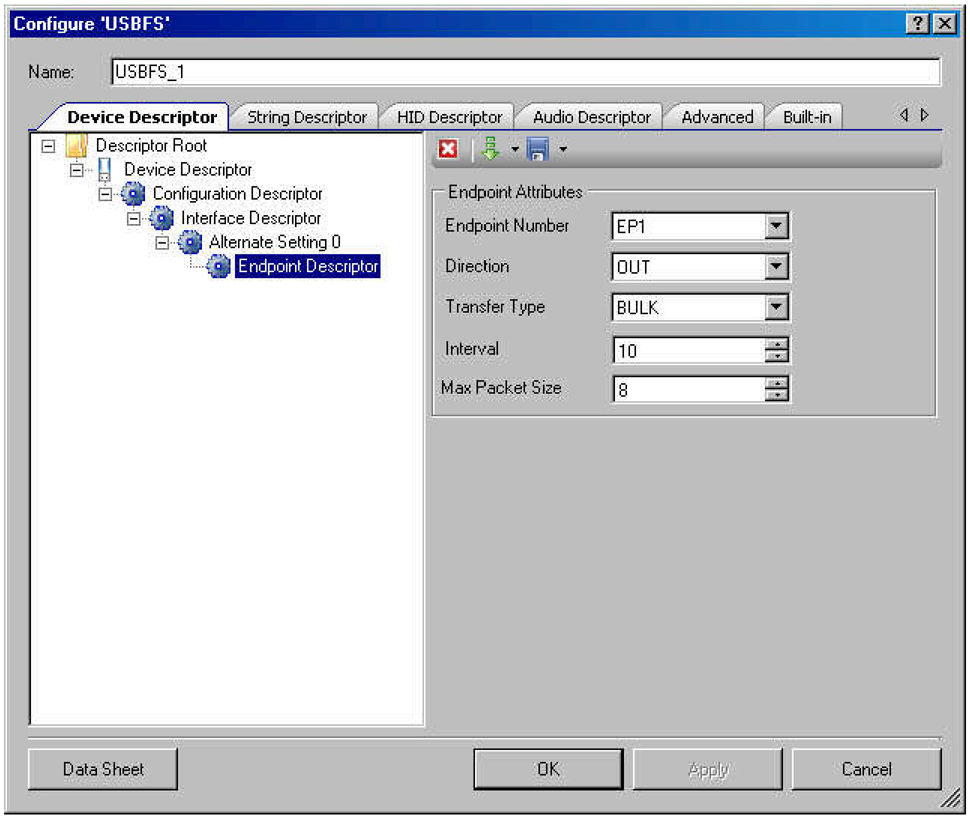
Device Attributes

Descriptor Type: Report
Country Code: Not Supported
HID Report: 3-Button Mouse

Device Attributes

- Descriptor Type – Constant name identifying type of class descriptor.
- Country Code – Numeric expression identifying country code of the localized hardware.
- HID Report – List of available Report Descriptors. Report descriptors are taken from the **HID Descriptor** tab. This field is required.

Endpoint Descriptor



Name: USBFS_1

Endpoint Attributes

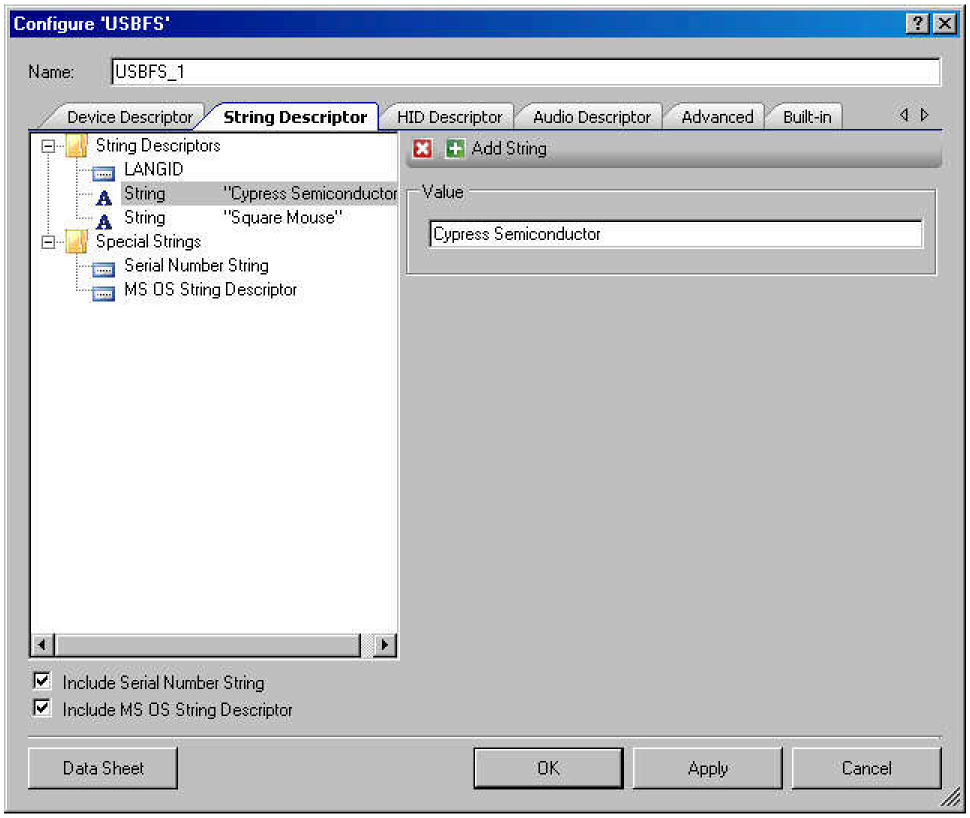
Endpoint Number	EP1
Direction	OUT
Transfer Type	BULK
Interval	10
Max Packet Size	8

Endpoint Attributes

- Endpoint Number
- Direction – Input or Output. USB transfers are host centric. Therefore IN refers to transfers to the host; OUT refers to transfers from the host.
- Transfer Type – Control, Interrupt, Bulk, or Isochronous Data transfers
- Interval (ms) – Polling interval specific to this endpoint. A full-speed endpoint can specify a desired period from 1 ms to 255 ms.
- Max Packet Size (bytes) – For a full speed device the Max Packet Size is 64 bytes for bulk or interrupt endpoints and 1023 bytes for isochronous endpoints.

String Descriptor Tab

String Descriptors


Configure 'USBFS'

Name: USBFS_1

String Descriptors
- LANGID
- String "Cypress Semiconductor
- String "Square Mouse"

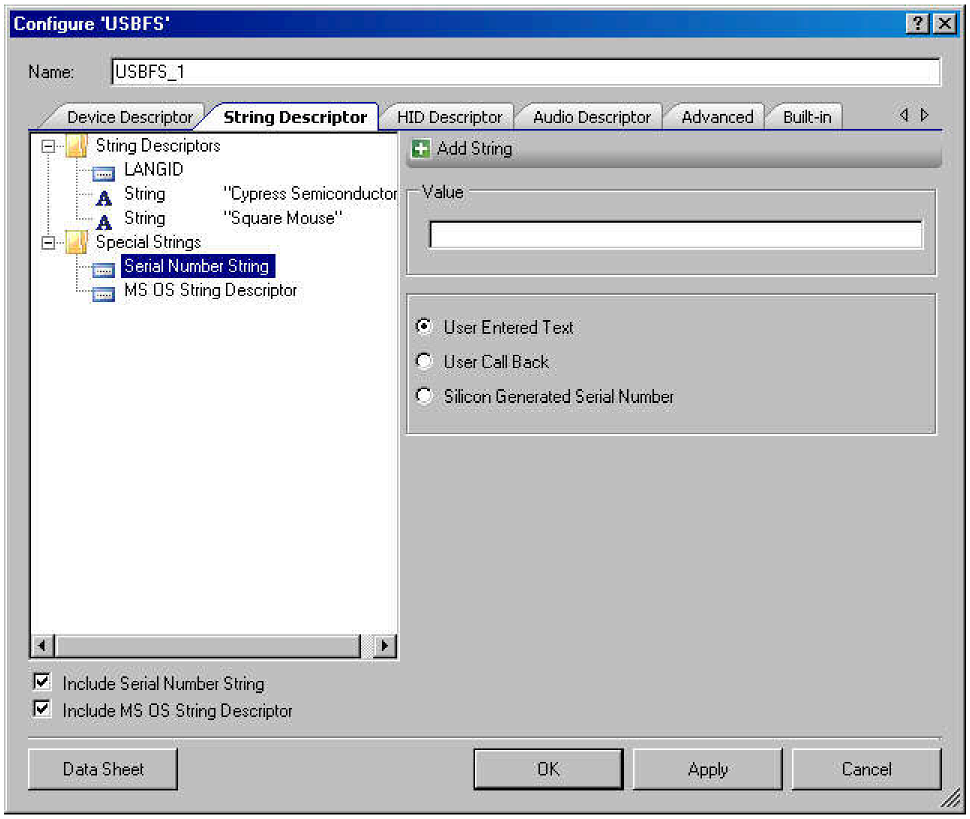
Special Strings
- Serial Number String
- MS OS String Descriptor

Value: Cypress Semiconductor

☑ Include Serial Number String
☑ Include MS OS String Descriptor

- LANGID – Language ID selection.
- String – Value of string descriptor.

Serial Number String



Configure 'USBFS'

Name: USBFS_1

Tabs: Device Descriptor, String Descriptor, HID Descriptor, Audio Descriptor, Advanced, Built-in

- String Descriptors
  - LANGID
  - String "Cypress Semiconductor
  - String "Square Mouse"
- Special Strings
  - Serial Number String
  - MS OS String Descriptor

Add String

Value:

(●) User Entered Text
( ) User Call Back
( ) Silicon Generated Serial Number

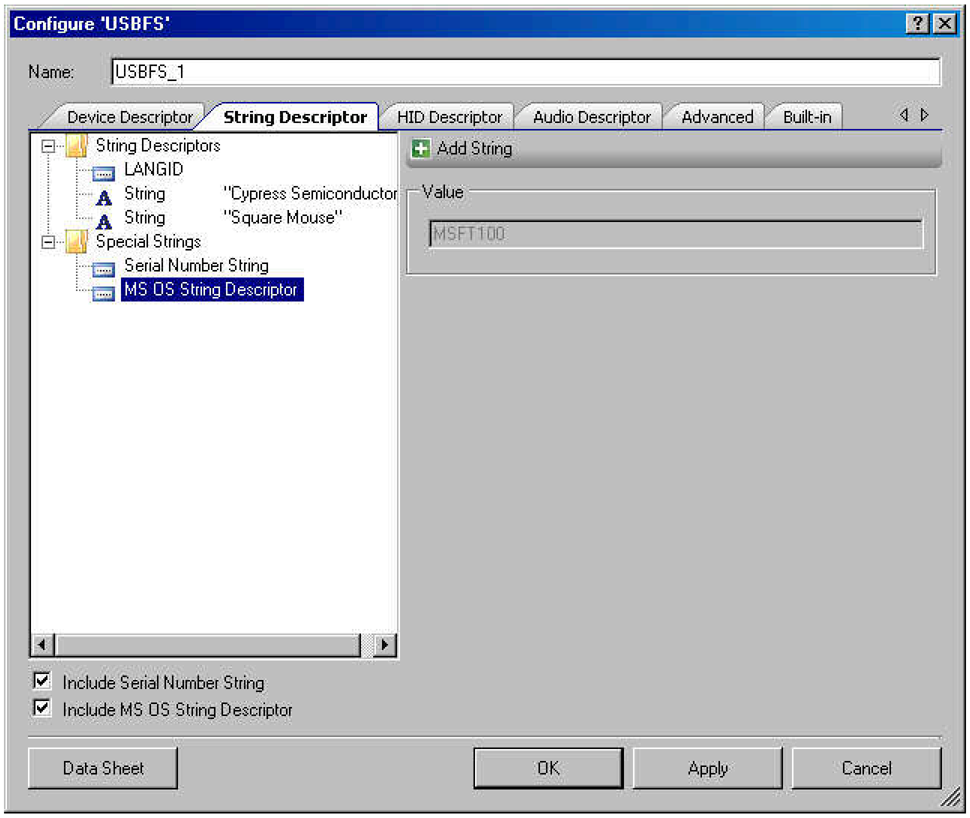
☑ Include Serial Number String
☑ Include MS OS String Descriptor

- Value – Default string.
- User Entered Text – Enables Value text box.
- User Call Back – USBFS_SerialNumString function sets pointer to use the user generated serial number string descriptor. The application firmware may supply the source of the USB device descriptor's serial number string during runtime.
- Silicon Generated Serial Number

PRELIMINARY

MS OS String Descriptor

Microsoft OS Descriptors provide a way for USB devices to supply additional configuration information to the latest Microsoft operating systems

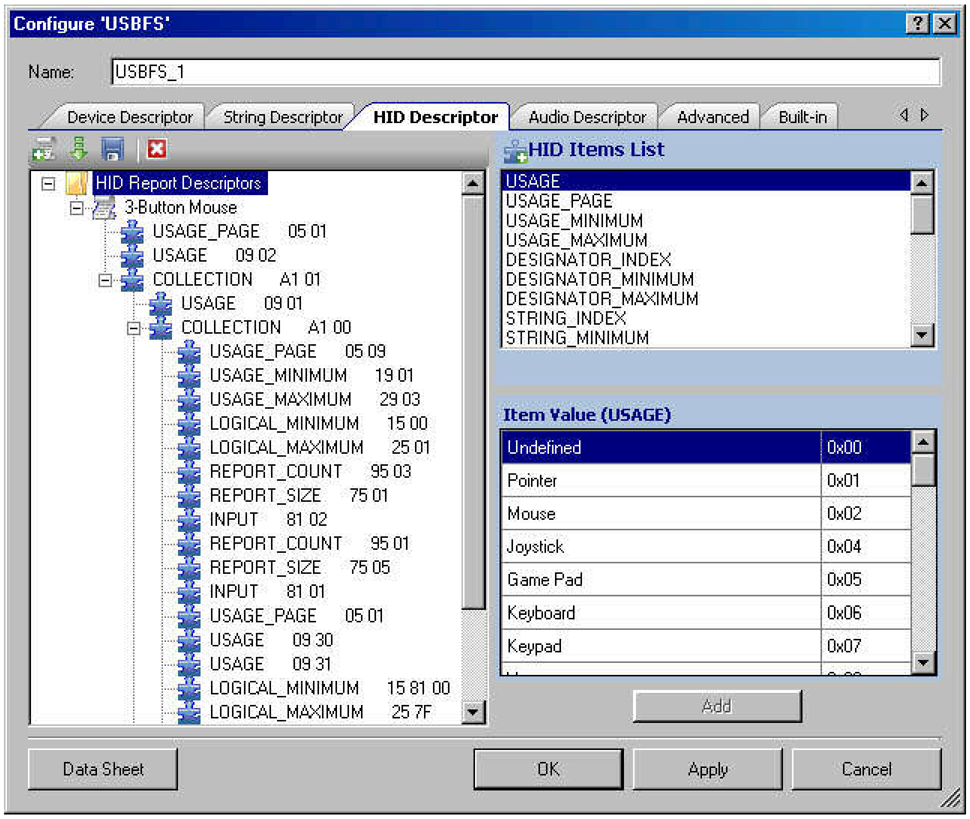


- Value – constant string "MSFT100"

HID Descriptor Tab

The **HID Descriptor** tab allows you to quickly build HID descriptors for your device. Use the **Add Report** button to add and configure HID Report Descriptors.

HID Descriptors



Name: USBFS_1

```
HID Report Descriptors
  3-Button Mouse
    USAGE_PAGE     05 01
    USAGE     09 02
    COLLECTION     A1 01
      USAGE     09 01
      COLLECTION     A1 00
        USAGE_PAGE     05 09
        USAGE_MINIMUM     19 01
        USAGE_MAXIMUM     29 03
        LOGICAL_MINIMUM     15 00
        LOGICAL_MAXIMUM     25 01
        REPORT_COUNT     95 03
        REPORT_SIZE     75 01
        INPUT     81 02
        REPORT_COUNT     95 01
        REPORT_SIZE     75 05
        INPUT     81 01
        USAGE_PAGE     05 01
        USAGE     09 30
        USAGE     09 31
        LOGICAL_MINIMUM     15 81 00
        LOGICAL_MAXIMUM     25 7F
```

```
USAGE
USAGE_PAGE
USAGE_MINIMUM
USAGE_MAXIMUM
DESIGNATOR_INDEX
DESIGNATOR_MINIMUM
DESIGNATOR_MAXIMUM
STRING_INDEX
STRING_MINIMUM
```

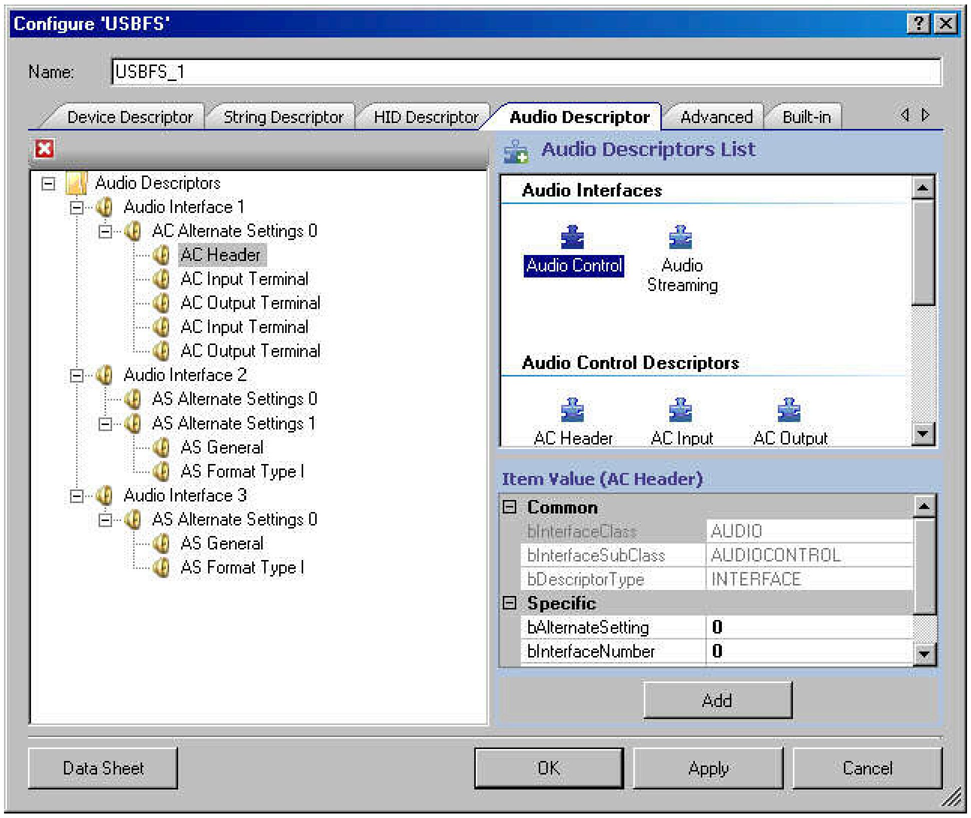
Undefined	0x00
Pointer	0x01
Mouse	0x02
Joystick	0x04
Game Pad	0x05
Keyboard	0x06
Keypad	0x07

- HID Items List – Items to add in the HID report.
- Item Value – Value of the item that is selected either in HID Items List or in the tree.

PRELIMINARY

Audio Descriptor Tab

The **Audio Descriptor** tab is used to add and configure audio interface descriptors.



To Add Audio Descriptors

1. Select the "Audio Descriptors" root item in the tree on the left.
2. Under **Audio Descriptors List** on the right, select either the "Audio Control" or "Audio Streaming" interface.
3. Under **Item Value**, enter bAlternateSetting and bInterfaceNumber values as appropriate. Other fields are optional.

   Note These values are set manually. By contrast, for the general interface descriptors, these values are set automatically.
4. Click **Add** and the descriptor is added to the tree on the left.

   You can rename the "Audio Interface x" title by selecting a node and then clicking on it.

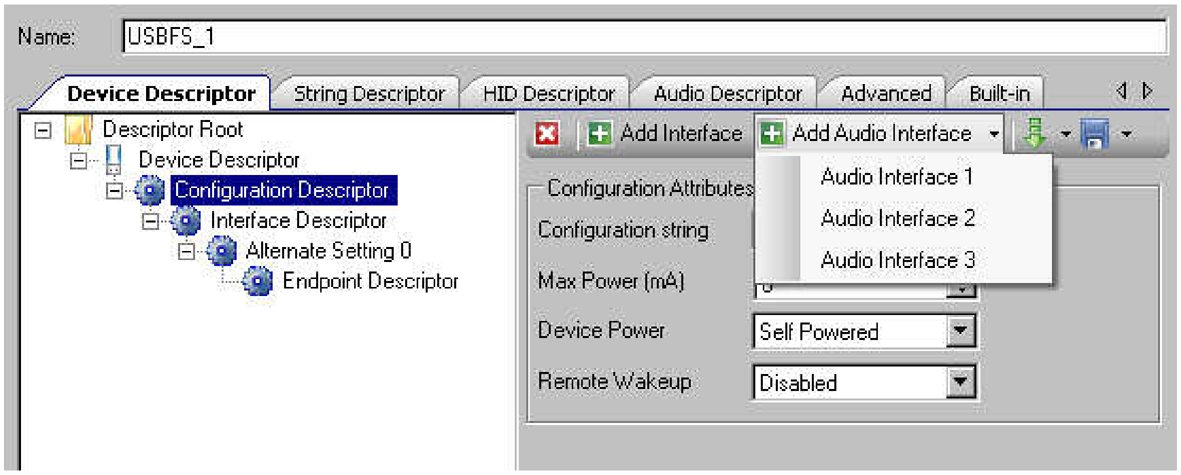
To Add Class-Specific Audio Control or Audio Streaming Interface Descriptors

1. Select the appropriate "AC Alternate Settings x" or "AS Alternate Settings x" item in the tree on the left.

2. Under the **Audio Descriptors List** on the right, select one of the items under "Audio Control Descriptors" or "Audio Streaming Descriptors" as appropriate.
3. Under **Item Value**, enter the appropriate values under "Specific."
4. Click **Add** and the descriptor is added to the tree on the left.

To add the configured audio interface descriptor to the Device Descriptor tree

1. Go to the **Device Descriptor** tab.
2. Select the Configuration Descriptor to which a new interface will belong.
3. Click the **Add Audio Interface** tool button, and select the appropriate item to add.

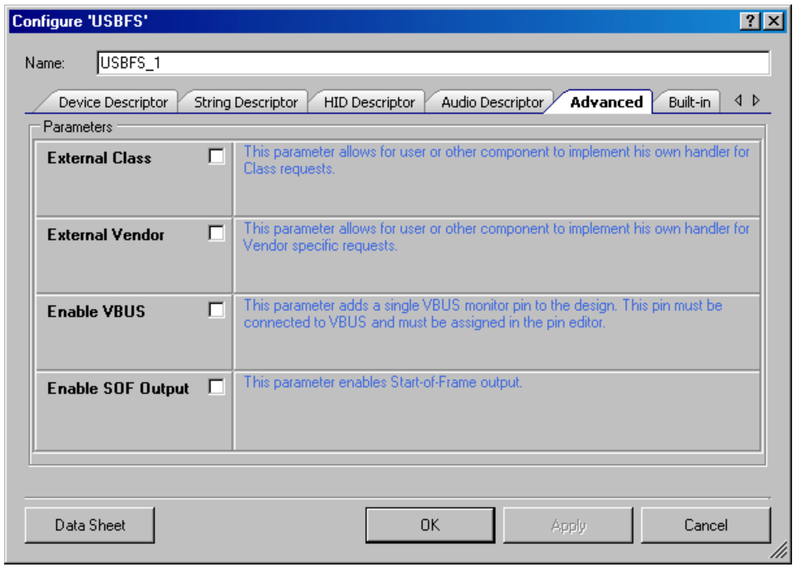


Audio interfaces will be grayed out in the **Device Descriptor** tab list because they can only be edited on the **Audio Descriptor** tab.

Endpoint descriptors could be added to the audio interfaces as usual.

Note Click **Apply** or **OK** to save the changes on the various tabs. If you click **Cancel** all the descriptors you added will not be saved.

Advanced Tab



External Class	☐	This parameter allows for user or other component to implement his own handler for Class requests.
External Vendor	☐	This parameter allows for user or other component to implement his own handler for Vendor specific requests.
Enable VBUS	☐	This parameter adds a single VBUS monitor pin to the design. This pin must be connected to VBUS and must be assigned in the pin editor.
Enable SOF Output	☐	This parameter enables Start-of-Frame output.

External Class

This parameter allows for the user firmware, or other components at the solutions level, to provide handling of the class requests. USBFS_DispatchClassRqst() function should be implemented if this parameter is enabled.

External Vendor

This parameter allows for the user firmware, or other components at the solutions level, to provide handling of the vendor specific requests. USBFS_HandleVendorRqst() function should be implemented if this parameter is enabled.

Enable VBUS Monitoring

The USB specification requires that no device shall supply current on VBUS at its upstream facing port at any time. To meet this requirement, the device must monitor for the presence or absence of VBUS and remove power from the D+/D- pull-up resistor if VBUS is absent.

For bus powered designs, power will obviously be removed when the USB cable is removed from a host; however, for self-powered designs it is imperative for proper operation and USB certification that your device complies with this requirement.

This parameter adds a single VBUS monitor pin to the design. This pin must be connected to the VBUS and must be assigned in the Pin Editor. See the USB Compliance for Self Powered Devices section for additional information.

Enable SOF output

This parameter enables Start-of-Frame output.

Placement

USB is implemented as a fixed function block.

Resources

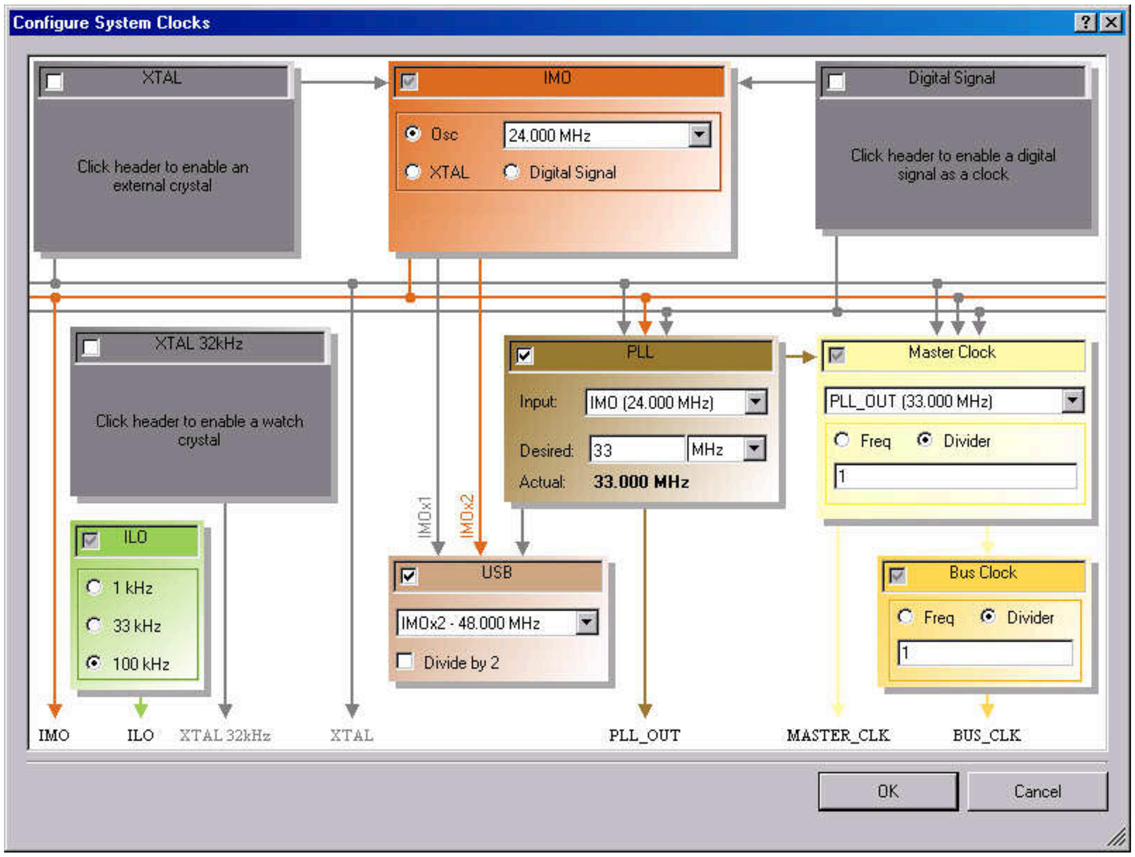
The USBFS uses the USB fixed function block.

Clock Settings

The USB hardware block requires system clocks to be configured through the PSoC Creator Design-Wide Resources Clock Editor. Clock settings have the following requirements when using the USBFS component:

- The USB Clock must be enabled.
- The ILO must be set to 100 KHz.
- If the selected device is PSoC 3 ES2 or PSoC 5, the Bus Clock cannot be slower than 33 MHz [for Master Clock, select PLL_OUT (33.000 MHz)].

There are different ways to configure the system clocks to comply with these requirements. The following shows one set of options you may use. Your design may require different settings.



Configure System Clocks

XTAL: Click header to enable an external crystal

IMO: [✓ Osc] XTAL Digital Signal — 24.000 MHz

Digital Signal: Click header to enable a digital signal as a clock

XTAL 32kHz: Click header to enable a watch crystal

ILO: 1 kHz 33 kHz [✓ 100 kHz]

PLL: Input: IMO (24.000 MHz) Desired: 33 MHz Actual: 33.000 MHz

Master Clock: PLL_OUT (33.000 MHz) Freq [✓ Divider] 1

USB: IMOx2 - 48.000 MHz Divide by 2 [ ]

Bus Clock: Freq [✓ Divider] 1

PRELIMINARY

Application Programming Interface

Application Programming Interface (API) routines allow you to configure the component using software. The following table lists and describes the interface to each function. The subsequent sections cover each function in more detail.

By default, PSoC Creator assigns the instance name "USBFS_1" to the first instance of a component in a given design. You can rename it to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "USBFS".

Basic USBFS Device APIs

Function	Description
USBFS_Start	Activate the component for use with the device and specific voltage mode.
USBFS_Init	Initialize component's hardware.
USBFS_InitComponent	Initialize component's global variables and initiate communication with Host by pull up D+ line.
USBFS_Stop	Disable component.
USBFS_GetConfiguration	Returns the currently assigned configuration. Returns 0 if the device is not configured.
USBFS_GetInterfaceSetting	Returns the current alternate setting for the specified interface.
USBFS_GetEPState	Returns the current state of the specified USBFS endpoint.
USBFS_GetEPAckState	Identifies whether ACK was set by returning a non-zero value.
USBFS_GetEPCount	Returns the current byte count from the specified USBFS endpoint.
USBFS_LoadInEP	Loads and enables the specified USBFS endpoint for an IN transfer.
USBFS_ReadOutEP	Reads the specified number of bytes from the Endpoint RAM and places it in the RAM array pointed to by pSrc. The function returns the number of bytes sent by the host.
USBFS_EnableOutEP	Enables the specified USB endpoint to accept OUT transfers
USBFS_DisableOutEP	Disables the specified USB endpoint to NAK OUT transfers
USBFS_SetPowerStatus	Sets the device to self powered or bus powered
USBFS_Force	Forces a J, K, or SE0 State on the USB D+/D- pins. Normally used for remote wakeup.
USBFS_SerialNumString	Provides the source of the USB device serial number string descriptor during runtime.

PRELIMINARY

Global Variables

Variable	Description
USBFS_initVar	Indicates whether the USBFS has been initialized. The variable is initialized to 0 and set to 1 the first time USBFS_Start() is called. This allows the component to restart without reinitialization in after the first call to the USBFS_Start() routine. If reinitialization of the component is required the variable should be set to 0 before the USBFS_Start() routine is called. Alternately, the USBFS can be reinitialized by calling the USBFS_Init() and USBFS_InitComponent() functions.
USBFS_device	Contains the started device number from the desired device descriptor set entered with the USBFS customizer.
USBFS_transferState	This variable used by the communication functions to handle current transfer state. Initialized to TRANS_STATE_IDLE in USBFS_InitComponent() API.
USBFS_configuration	Contains current configuration number which is set by the Host using SET_CONFIGURATION request. This variable is initialized to zero in USBFS_InitComponent() API, returns to the application level by the USBFS_GetConfiguration() API._
USBFS_deviceAddress	Contains current device address. This variable is initialized to zero in USBFS_InitComponent() API. Host starts to communicate to device with address 0 and then set it to whatever value using SET_ADDRESS request.
USBFS_deviceStatus	This is two bit variable which contain power status in first bit (DEVICE_STATUS_BUS_POWERED or DEVICE_STATUS_SELF_POWERED) and remote wakeup status (DEVICE_STATUS_REMOTE_WAKEUP) in second bit. This variable is initialized to zero in USBFS_InitComponent() API, configured by the USBFS_SetPowerStatus() API.

void USBFS_Start (uint8 device, uint8 mode)

Description: Performs all required initialization for USBFS Component.

Parameters: (uint8) device: Contains the device number from the desired device descriptor set entered with the USBFS customizer.

(uint8) mode: The operating voltage. This determines whether the voltage regulator is enabled for 5V operation or if pass through mode is used for 3.3 V operation. Symbolic names and their associated values are given in the following table.

Power Setting	Notes
USBFS_3V_OPERATION	Disable voltage regulator and pass-thru Vcc for pull-up
USBFS_5V_OPERATION	Enable voltage regulator and use regulator for pull-up
USBFS_DWR_VDDD_OPERATION	Enable or Disable voltage regulator depend on Vddd Voltage configuration in DWR.

Return Value: None

Side Effects: None

void USBFS_Init (void)

Description: Initialize component's hardware.

Parameters: None

Return Value: None

Side Effects: None

void USBFS_InitComponent (uint8 device, uint8 mode)

Description: Initialize component's global variables and initiate communication with Host by pull up D+ line.

Parameters: (uint8) device: Contains the device number from the desired device descriptor set entered with the USBFS customizer.

(uint8) mode: The operating voltage. This determines whether the voltage regulator is enabled for 5V operation or if pass through mode is used for 3.3V operation. Symbolic names and their associated values are given in the following table.

Power Setting	Notes
USBFS_3V_OPERATION	Disable voltage regulator and pass-thru Vcc for pull-up
USBFS_5V_OPERATION	Enable voltage regulator and use regulator for pull-up
USBFS_DWR_VDDD_OPERATION	Enable or Disable voltage regulator depend on Vddd Voltage configuration in DWR.

Return Value: None

Side Effects: This function called from the Reset ISR to initialize communication.

void USBFS_Stop (void)

Description: Performs all necessary shutdown task required for the USBFS Component.

Parameters: None

Return Value: None

Side Effects: None

uint8 USBFS_GetConfiguration(void)

Description: Gets the current configuration of the USB device.

Parameters: None

Return Value: uint8: Returns the currently assigned configuration. Returns 0 if the device is not configured.

Side Effects: None

uint8 USBFS_GetInterfaceSetting(uint8 interfaceNumber)

Description: Gets the current alternate setting for the specified interface.

Parameters: uint8 interfaceNumber: Interface number

Return Value: uint8: Returns the current alternate setting for the specified interface.

Side Effects: None

uint8 USBFS_GetEPState(uint8 epNumber)

Description: Returns the state of the requested endpoint.

Parameters: (uint8) epNumber: The data endpoint number.

Return Value: (uint8) Returns the current state of the specified USBFS endpoint. Symbolic names provided, and their associated values are given in the following table. Use these constants whenever you write code to change the state of the endpoints such as ISR code to handle data sent or received.

Return Value	Description
USBFS_NO_EVENT_PENDING	Indicates that the endpoint is awaiting SIE action
USBFS_EVENT_PENDING	Indicates that the endpoint is awaiting CPU action
USBFS_NO_EVENT_ALLOWED	Indicates that the endpoint is locked from access
USBFS_IN_BUFFER_FULL	The IN endpoint is loaded and the mode is set to ACK IN
USBFS_IN_BUFFER_EMPTY	An IN transaction occurred and more data can be loaded
USBFS_OUT_BUFFER_EMPTY	The OUT endpoint is set to ACK OUT and is waiting for data
USBFS_OUT_BUFFER_FULL	An OUT transaction has occurred and data can be read

Side Effects: None

uint8 USBFS_GetEPAckState(uint8 epNumber)

Description: Determines whether or not an ACK transaction occurred on this endpoint by reading the ACK bit in the control register of the endpoint. This function does not clear the ACK bit.

Parameters: (uint8) epNumber: Contains the data endpoint number.

Return Value: (uint8): If an ACKed transaction occurred then this function returns a non-zero value. Otherwise a zero is returned.

Side Effects: None

uint16 USBFS_GetEPCount(uint8 epNumber)

Description: Returns the transfer count for the requested endpoint. The value from the count registers includes 2 counts for the two byte checksum of the packet. This function subtracts the two counts.

Parameters: (uint8) epNumber: Contains the data endpoint number.

Return Value: (uint16): Returns the current byte count from the specified USBFS endpoint or 0 for an invalid endpoint.

Side Effects: None

void USBFS_LoadInEP(uint8 epNumber, uint8 *pData, uint16 length)

Description: Loads and enables the specified USB data endpoint for an IN interrupt or bulk transfer.

Parameters: (uint8) epNumber: Contains the data endpoint number.

(uint8) *pData: A pointer to a data array from which the data for the endpoint space is loaded.

(uint16) length: The number of bytes to transfer from the array and then send as a result of an IN request. Valid values are between 0 and 512.

Return Value: None

Side Effects: None

uint16 USBFS_ReadOutEP(uint8 epNumber, uint8 *pData, uint16 length)

Description: Moves the specified number of bytes from endpoint RAM to data RAM. The number of bytes actually transferred from endpoint RAM to data RAM is the lesser of the actual number of bytes sent by the host or the number of bytes requested by the wCount parameter.

Parameters: (uint8) epNumber: Contains the data endpoint number.

(uint8) *pData: A pointer to a data array from which the data for the endpoint space is loaded.

(uint16) length: The number of bytes to transfer from the USB OUT endpoint and loads it into data array. Valid values are between 0 and 512. The function moves fewer than the requested number of bytes if the host sends fewer bytes than requested.

Return Value: (uint16): Number of bytes received

Side Effects: None

void USBFS_EnableOutEP(uint8 epNumber)

Description: Enables the specified endpoint for OUT bulk or interrupt transfers. Do not call this function for IN endpoints.

Parameters: (uint8) epNumber: Contains the data endpoint number.

Return Value: None

Side Effects: None

void USBFS_DisableOutEP(uint8 epNumber)

Description: Disables the specified USBFS OUT endpoint. Do not call this function for IN endpoints.

Parameters: (uint8) epNumber: Contains the data endpoint number.

Return Value: None

Side Effects: None

void USBFS_SetPowerStatus(uint8 powerStatus)

Description: Sets the current power status. The device will reply to USB GET_STATUS requests based on this value. This allows the device to properly report its status for USB Chapter 9 compliance. Devices may change their power source from self powered to bus powered at any time and report their current power source as part of the device status. You should call this function any time your device changes from self powered to bus powered or vice versa, and set the status appropriately.

Parameters: (uint8) powerStatus: Contains the desired power status, one for self powered or zero for bus powered. Symbolic names are provided and their associated values are given here:

Power Status	Description
USBFS_DEVICE_STATUS_BUS_POWERED	Set the device to bus powered.
USBFS_DEVICE_STATUS_SELF_POWERED	Set the device to self powered.

Return Value: None

Side Effects: None

void USBFS_Force(uint8 state)

Description: Forces a USB J, K, or SE0 state on the D+/D- lines. This function provides the necessary mechanism for a USB device application to perform a USB Remote Wakeup. For more information, refer to the USB 2.0 Specification for details on Suspend and Resume.

Parameters: (uint8) state: A byte indicating which of the four bus states to enable. Symbolic names provided, and their associated values are listed here:

State	Description
USBFS_FORCE_SE0	Force a Single Ended 0 onto the D+/D- lines
USBFS_FORCE_J	Force a J State onto the D+/D- lines
USBFS_FORCE_K	Force a K State onto the D+/D- lines
USBFS_FORCE_NONE	Return bus to SIE control

Return Value: None

Side Effects: None

PRELIMINARY

void USBFS_SerialNumString(uint8 *snString)

Description: This function available only when the "User Call Back" option in the "Serial Number String" descriptor properties is selected. Application firmware may provide the source of the USB device serial number string descriptor during runtime. Default string will be used if the application firmware does not use this function or sets the wrong string descriptor.

Parameters: (uint8) *snString: pointer to user defined string descriptor. String descriptor should meet the Universal Serial Bus Specification revision 2.0 chapter 9.6.7

Return Value: None

Side Effects: None

Human Interface Device (HID) Class Support

Function	Description
USBFS_UpdateHIDTimer	Updates the HID Report timer for the specified interface and returns 1 if the timer expired and 0 if not. If the timer expired, it reloads the timer.
USBFS_GetProtocol	Returns the protocol for the specified interface

uint8 USBFS_UpdateHIDTimer(uint8 interface)

Description: Updates the HID Report idle timer and returns the status. Reloads the timer if it expires.

Parameters: (uint8) interface: Contains the interface number.

Return Value: (uint8): Returns the state of the HID timer. Symbolic names are provided and their associated values are given here:

Return Value	Notes
USBFS_IDLE_TIMER_EXPIRED	The timer expired.
USBFS_IDLE_TIMER_RUNNING	The timer is running.
USBFS_IDLE_TIMER_IDEFINITE	Returned if the report is sent when data or state changes.

Side Effects: None

uint8 USBFS_GetProtocol(uint8 interface)

Description: Returns the HID protocol value for the selected interface.

Parameters: (uint8) interface: Contains the interface number.

Return Value: (uint8): Returns the protocol value.

Side Effects: None

Boot Loader Support

The USBFS component could be used as a communication component for the Bootloader. The following configurations should be used to support communication protocol from an external system to the Bootloader:

- Endpoint Number: EP1, Direction: OUT, Transfer Type – INT, Max Packet Size: 64
- Endpoint Number: EP2, Direction: IN, Transfer Type – INT, Max Packet Size: 64

Full recommended configurations are stored in the template file (*bootloader.root.xml*). Select Descriptor Root on the Device Descriptor tree, click the Import button, browse to the following directory and open *bootloader.root.xml* file.

<INSTALL>\psoc\content\cycomponentlibrary\CyComponentLibrary.cylib\USBFS_v1_50\Custom \template\

See more information about Bootloader in System Reference Guide.

The USBFS Component provides a set of API functions for the Bootloader usage.

Function	Description
CyBtldrCommStart	Performs all required initialization for USBFS Component, waits on the enumeration and enables the communication.
CyBtldrCommStop	Calls the USBFS_Stop function.
CyBtldrCommReset	Resets the receive and transmit communication buffers.
CyBtldrCommWrite	Allows the caller to write data to the boot loader host. The function will handle polling to allow a block of data to be completely sent to the host device.
CyBtldrCommRead	Allows the caller to read data from the boot loader host. The function will handle polling to allow a block of data to be completely received from the host device.

void CyBtldrCommStart (void)

Description:	Performs all required initialization for USBFS component, waits on the enumeration and enables the communication.
Parameters:	None
Return Value:	None
Side Effects:	This function starts the USBFS with 3 V operation.

PRELIMINARY

void CyBtldrCommStop (void)

Description: Performs all necessary shutdown task required for the USBFS component.

Parameters: None

Return Value: None

Side Effects: Calls the USBFS_Stop function.

void CyBtldrCommReset (void)

Description: Resets the receive and transmit communication buffers.

Parameters: None

Return Value: None

Side Effects: None

cystatus CyBtldrCommWrite(uint8 *data, uint16 size, uint16 *count, uint8 timeOut)

Description: Allows the caller to write data to the boot loader host. The function will handle polling to allow a block of data to be completely sent to the host device.

Parameters: (uint8) *data: A pointer to the block of data to send to the device.

(uint16) size: The number of bytes to write.

(uint16) *count: Pointer to an unsigned short variable to write the number of bytes actually written.

(uint8) timeout: Number of units to wait before returning because of a timeout.

Return Value: (cystatus): Returns 1 if no problem was encountered or returns the value that best describes the problem.

Side Effects: None

cystatus CyBtldrCommRead(uint8 *data, uint16 size, uint16 *count, uint8 timeOut)

Description: Allows the caller to read data from the boot loader host. The function will handle polling to allow a block of data to be completely received from the host device.

Parameters: (uint8) *data: A pointer to the area to store the block of data received from the device.

(uint16) size: The number of bytes to read.

(uint16) *count: Pointer to an unsigned short variable to write the number of bytes actually read.

(uint8) timeOut: Number of units to wait before returning because of a timeout.

Return Value: (cystatus): Returns 1 if no problem was encountered or returns the value that best describes the problem.

Side Effects: None

USB Suspend, Resume, and Remote Wakeup

The USBFS Component supports USB Suspend, Resume, and Remote Wakeup. Since these features are tightly coupled into the user application, the USBFS Component provides a set of API functions.

Function	Description
USBFS_CheckActivity	Checks and clears the USB bus activity flag. Returns 1 if the USB was active since the last check, otherwise returns 0.
USBFS_Suspend	This function disables the USBFS block and prepare for power down mode.
USBFS_Resume	This function enables the USBFS block after power down mode.
USBFS_RWUEnabled	This function returns current remote wake up status.

uint8 USBFS_CheckActivity(void)

Description: Returns the activity status of the bus. Clears the status hardware to provide fresh activity status on the next call of this routine.
This function provides a means to check if any USB bus activity occurred. The application uses the function to determine if the conditions to enter USB Suspend were met.

Parameters: None

Return Value: (uint8) cystatus: Standard API return values.

Return Value	Description
1	If bus activity was detected since the last call to this function
0	If bus activity was not detected since the last call to this function

Side Effects: None

PRELIMINARY

void USBFS_Suspend(void)

Description: This function disables the USBFS block and prepare for power down mode. Should be called just prior to entering sleep.
Once the conditions to enter USB suspend are met, the application takes appropriate steps to reduce current consumption to meet suspend current requirements. To put the USB SIE and transceiver into power down mode, the application calls the USBFS_Suspend() API function and the USBFS_CheckActivity() API to detect USB activity. This function disables the USBFS block, but maintains the current USB address (in the USBCR register). The device uses the sleep feature to reduce power consumption.

Parameters: void

Return Value: void

Side Effects:

void USBFS_Resume(void)

Description: This function enables the USBFS block after power down mode. Should be called just after awaking from sleep.
While the device is suspended, it periodically checks to determine if the conditions to leave the suspended state were met. One way to check resume conditions is to use the sleep timer to periodically wake the device. If the resume conditions were met, the application calls the USBFS_Resume API function. This function enables the USBFS SIE and Transceiver, bringing them out of power down mode. It does not change the USB address field of the USBCR register, maintaining the USB address previously assigned by the host.

Parameters: void

Return Value: void

Side Effects:

uint8 USBFS_RWUEnabled(void)

Description: This function returns current remote wake up status.
If the device supports remote wakeup, the application is able to determine if the host enabled remote wakeup with the USBFS_RWUEnabled API function. When the device is suspended and it determines the conditions to initiate a remote wakeup are met, the application uses the USBFS_Force API function to force the appropriate J and K states onto the USB Bus, signaling a remote wakeup.

Parameters: void

Return Value: TRUE - Remote Wake Up Enabled
FALSE - Remote Wake Up Disabled

Side Effects:

Sample Firmware Source Code

The following are C language examples demonstrating the basic functionality of the USBFS component. These examples assume the component has been placed in a design with the default name "USBFS_1."

Note If you rename your component you must also edit the example code as appropriate to match the component name you specify.

Generic USB Bulk Wraparound Transfer Example

The following example enumerates as a Vendor-Specific device with 1 Bulk IN and 1 Bulk OUT endpoints. The code simply wraps the OUT data, coming from the host, back to the host on a subsequent IN.

```
#include <device.h>

#define IN_EP       0x01
#define OUT_EP      0x02
#define BUF_SIZE    0x40

uint8 buffer[BUF_SIZE];
uint8 length;

void main()
{
    CYGlobalIntEnable;                          /* Enable Global Interrupts */
    USBFS_1_Start(0, USBFS_1_3V_OPERATION);     /* Start USBFS Operation with */
                                                /* 3V operation */

    while(!USBFS_1_GetConfiguration());         /*Wait for Device to enumerate */

    /*Enumeration is completed enable OUT endpoint for receive data from Host.*/
    USBFS_1_EnableOutEP(OUT_EP);

    while(1)
    {
        /* Wait for data received */
        while(USBFS_1_GetEPState(OUT_EP) != USBFS_1_OUT_BUFFER_FULL)
        /* Read received bytes count */
        length = USBFS_1_GetEPCount(OUT_EP);
        /* Unload the OUT buffer */
        USBFS_1_ReadOutEP(OUT_EP, &buffer[0], length);
        /* Check for IN buffer is empty */
        while(USBFS_1_GetEPState(IN_EP) != USBFS_1_IN_BUFFER_EMPTY);
    }
}
```

PRELIMINARY

USBFS Setup Corresponding to the Example Code

The following are the minimum steps of configuring the USBFS component to correspond with the example code.

- Observe the default **Device Attributes**: Vendor ID – 0x4B4, Product ID – 0x8051.
- Edit the Endpoint Descriptor:
  - Select **Direction** – IN for EP1, **Transfer Type** – BULK, **Max Packet Size** – 64.
  - Add second Endpoint EP2 and select **Direction** – OUT, **Transfer Type** – BULK, **Max Packet Size** – 64.
- Click **OK** to save the USB descriptor information.

The following are the steps to test this example.

- Install Cypress SuiteUSB Development tools.
- Update the *cyusb.inf* file in the Driver subdirectory with the correct VID/PID numbers: VID_04B4&PID_8051.
- Build and program this project to DVK1 board. Select 3.3 V in SW3, plug-in power and connect the DVK board to a PC by a USB cable.
- Select the *cyusb.inf* file and *cyusb.sys* as the driver for this example once the windows ask for it.
- Open the "USB Console" application (installed as a part of SuiteUSB), select "Cypress USB Generic" device and send test data to the OUT endpoint (EP2), then initialize reading IN endpoint (EP1) and observe same data received.

HID Device Example

The following C code shows you how to use the USBFS component in a simple HID application. Once connected to a PC host, the device enumerates as a 3-button mouse. When the code is run, the mouse cursor zigzags from right to left. This code illustrates the how the USBFS customizer configures the component.

```
#include <device.h>

uint8 abMouseData[3] = {0,0,0};
uint8 snString[16]={0x0E,0x03,'F',0,'W',0,'S',0,'N',0,'0',0,'1',0};

uint8 i = 0;
void main()
{
    CYGlobalIntEnable;                          /*Enable Global Interrupts*/
    /* Set user defined Serial Number string */
    USBFS_1_SerialNumString(&snString[0]);
    USBFS_1_Start(0, USBFS_1_3V_OPERATION);     /*Start USBFS Operation/device 0*/
                                                /*and with 3V operation*/
```

```
        while(!USBFS_1_GetConfiguration());       /*Wait for Device to enumerate*/

        while(1)
        {
          while(!USBFS_1_GetEPAckState(1));  /*Wait for ACK before loading data*/

          if(i==128)                        /*When our count hits 128*/
          {
            abMouseData[1] = 0x05;          /*Start moving the mouse to the right*/
          }
          else if(i==255)                   /*When our counts hits 255*/
          {
            abMouseData[1] = 0xFB;          /*Start moving the mouse to the left*/
          }
          i++;
        }
    }
```

USBFS Setup Corresponding to the Example Code

The following are the basic walk-through steps of configuring the USBFS component to correspond with the example code.

- Click the **HID Descriptor** tab, select **the Import Report** command, and browse to the following directory:

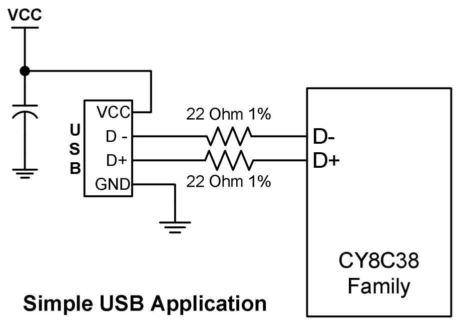
  <INSTALL>\psoc\content\cycomponentlibrary\CyComponentLibrary.cylib\USBFS_v1_50\Custom\template\

- Select the 3-button mouse template file (*3ButtonMouse.hid.xml*) and click **Open** button. Various fields of information will be completed for this tab.
- Click the **String Descriptor** tab, select the **Add String** command twice and complete string values with Manufacturer and Product names.
- Select the User Call Back value for Serial Number String.
- Click the **Device Descriptor** tab and select Device Descriptor in Descriptor Root tree.
- Edit the device attributes: Vendor ID, Product ID, and select strings.
- Click **Alternate Settings** in Descriptor Root tree.
- Edit the interface attributes: select HID for the Class field.
- Click **HID Class Descriptor** in the Descriptor Root tree and select the 3 button mouse for the HID Report field.
- Click **HID Class Descriptor** in Descriptor Root tree and select Direction - IN, Transfer Type - INT.
- Click **OK** to save the USB descriptor information.

PRELIMINARY

Functional Description

The following diagram shows a simple bus-powered USB application with the D+ and D- pins from the PSoC device.



USB Compliance

USB drivers may present various bus conditions to the device, including Bus Resets, and different timing requirements. Not all of these can be correctly illustrated in the examples provided. It is your responsibility to design applications that conform to the USB spec.

USB Compliance for Self Powered Devices

If the device that you are creating will be self-powered, you must connect a GPIO pin to VBUS through a resistive network and write firmware to monitor the status of the GPIO. You can use the USBFS_Start() and USBFS_Stop() API routines to control the D+ and D- pin pull-ups. The pull-up resistor does not supply power to the data line until you call USBFS_Start(). USBFS_Stop() disconnects the pull-up resistor from the data pin.

The device responds to GET_STATUS requests based on the status set with the USBFS_SetPowerStatus() function. To set the correct status, USBFS_SetPowerStatus() should be called at least once if your device is configured as self-powered. You should also call the USBFS_SetPowerStatus() function any time your device changes status.

USB Standard Device Requests

This section describes the requests supported by the USBFS component. If a request is not supported the USBFS component responds with a STALL, indicating a request error.

Standard Device Request	USB Component Support Description	USB 2.0 Spec Section
CLEAR_FEATURE	Device:	9.4.1
	Interface: not supported.	
	Endpoint	

PRELIMINARY

Standard Device Request	USB Component Support Description	USB 2.0 Spec Section
GET_CONFIGURATION	Returns the current device configuration value.	9.4.2
GET_DESCRIPTOR	Returns the specified descriptor.	9.4.3
GET_INTERFACE	Returns the selected alternate interface setting for the specified interface.	9.4.4
GET_STATUS	Device:	9.4.5
	Interface:	
	Endpoint:	
SET_ADDRESS	Sets the device address for all future device accesses.	9.4.6
SET_CONFIGURATION	Sets the device configuration.	9.4.7
SET_DESCRIPTOR	This optional request is not supported.	9.4.8
SET_FEATURE	Device: DEVICE_REMOTE_WAKEUP support is selected by the bRemoteWakeUp Component Parameter. TEST_MODE is not supported.	9.4.9
	Interface: Not supported.	
	Endpoint: The specified Endpoint is halted.	
SET_INTERFACE	This request allows the host to select an alternate setting for the specified interface..	9.4.10
SYNCH_FRAME	Not supported. Future implementations of the Component will add support to this request to enable Isochronous transfers with repeating frame patterns.	9.4.11

HID Class Request

Class Request	USBFS Component Support Description	Device Class Definition for HID - Section
GET_REPORT	Allows the host to receive a report by way of the Control pipe.	7.2.1
GET_IDLE	Reads the current idle rate for a particular Input report.	7.2.3
GET_PROTOCOL	Reads which protocol is currently active (either the boot or the report protocol).	7.2.5
SET_REPORT	Allows the host to send a report to the device, possibly setting the state of input, output, or feature controls.	7.2.2

PRELIMINARY

Class Request	USBFS Component Support Description	Device Class Definition for HID - Section
SET_IDLE	Silences a particular report on the Interrupt In pipe until a new event occurs or the specified amount of time passes.	7.2.4
SET_PROTOCOL	Switches between the boot protocol and the report protocol (or vice versa).	7.2.6

DC and AC Electrical Characteristics

Not applicable

Component Changes

This section lists the major changes in the component from the previous version.

Version	Description of Changes	Reason for Changes / Impact
1.50.a	Made datasheet change log cumulative	Customer convenience.
1.50	USB Suspend, Resume, and Remote Wakeup functionality has been added.	USB device should support suspend and resume functionality.
	Most APIs renamed to remove foreign notation, old names are supported for backward compatibility.	To comply with corporate coding standards.
	GET_INTERFACE/SET_INTERFACE requests support has been added.	A device must support the GetInterface / SetInterface requests if it has alternate settings for that interface.
	Specific APIs were integrated to support the boot loader: CyBtldrCommStart, CyBtldrCommStop, CyBtldrCommReset, CyBtldrCommWrite, CyBtldrCommRead.	USB could be used as a communication component for the Boot Loader with this feature.
	Added generic USB Bulk Wraparound Transfer example to data sheet.	Described generic USB usage for user.
	Added the extern_cls and extern_vnd parameters to the Advanced tab of the Configure dialog.	These parameters enable other components at the solutions level, to provide their handling of Vendor and Class requests themselves.
	Restriction has been added to DMA w/Manual Memory Management section.	This restriction shows how to properly use Mode 2/3 transfers.
	'Advanced' tab layout modified.	The data grid was replaced with check boxes with information about each parameter. This was done to improve usability.

Version	Description of Changes	Reason for Changes / Impact
	Added Audio Descriptors tab to the Configure dialog.	This allows you to add and configure audio descriptors for your component.
	Removed SOF ISR enable/disable from Start/Stop APIs.	SOF interrupts occur each 1ms, but was not used by the component. If application requires this interrupt it can be enabled by calling: `CyIntEnable(USBFS_SOF_VECT_NUM);`
1.30.b	Added information to the component that advertizes its compatibility with silicon revisions.	The tool reports an error/warning if the component is used on incompatible silicon. If this happens, update to a revision that supports your target device.
1.30.a	Moved local parameters to formal parameter list.	To address a defect that existed in PSoC Creator v1.0 Beta 4.1 and earlier, the component was updated so that it could continue to be used in newer versions of the tool. This component used local parameters, which are not exposed to the user, to do background calculations on user input. These parameters have been changed to formal parameters which are visible, but un-editable. There are no functional changes to the component but the affected parameters are now visible in the "expression view" of the customizer dialog.
1.30	Updated the Configure dialog and data sheet.	Added the Enable SOF Output parameter to the Advanced tab of the Configure dialog. Updated the USBFS_ReadOutEP function in the data sheet to reflect the correct return value.
1.20.b	Added information to the component that advertizes its compatibility with silicon revisions.	The tool reports an error/warning if the component is used on incompatible silicon. If this happens, update to a revision that supports your target device.
1.20.a	Moved local parameters to formal parameter list.	To address a defect that existed in PSoC Creator v1.0 Beta 4.1 and earlier, the component was updated so that it could continue to be used in newer versions of the tool. This component used local parameters, which are not exposed to the user, to do background calculations on user input. These parameters have been changed to formal parameters which are visible, but un-editable. There are no functional changes to the component but the affected parameters are now visible in the "expression view" of the customizer dialog.
1.10.b	Added information to the component that advertizes its compatibility with silicon revisions.	The tool reports an error/warning if the component is used on incompatible silicon. If this happens, update to a revision that supports your target device.
1.10.a	Moved local parameters to formal parameter list.	To address a defect that existed in PSoC Creator v1.0 Beta 4.1 and earlier, the component was updated so that it could continue to be used in newer versions of the tool. This component used local parameters, which are not exposed to the user, to do background calculations on user input. These parameters have been changed to formal parameters which are visible, but un-editable. There are no functional changes to the component but the affected parameters are now visible in the "expression view" of the customizer dialog.