USB Mass Storage Class on an Embedded Host

Author: *Kim Otten*
Microchip Technology Inc.

INTRODUCTION

With the introduction of Microchip's microcontrollers with the USB OTG peripheral, microcontroller applications can easily support USB Embedded Host functionality. One of the most common uses of this capability is to interface to mass storage devices, such as USB Flash Drives and memory card readers. These devices utilize the USB Mass Storage Class.

USB Mass Storage Class

Overview

Of the four transfer types supported by USB, the one most suitable for large data transfers is Bulk. Bulk transfers use the USB bandwidth efficiently, in that they utilize all of the remaining bandwidth in a frame after Control, Interrupt and Isochronous transfers are complete. They are not constrained to only a certain number of bytes per frame. They also incorporate error checking, so the data is ensured to be accurate. The exact amount of time available for a bulk transfer will depend on the amount of other traffic that is on the bus. If several other transfers must also be performed, there may be very little bandwidth available for Bulk transfers in a frame. Therefore, Bulk transfers should be used only for non-time critical operations.

The class, subclass and protocol designators for a Mass Storage Device are not contained in the bDeviceClass, bDeviceSubClass and bDeviceProtocol fields of the Device Descriptor. Instead, these fields are all set to 0x00, and the designators are specified in the bInterfaceClass, bInterfaceSubClass and bInterfaceProtocol fields of the Interface Descriptor. The most common configuration for USB Mass Storage devices is:

- bInterfaceClass - 0x08 (Mass Storage Class)
- bInterfaceSubClass - 0x06 (SCSI Primary Command-2 (SPC-2))
- bInterfaceProtocol - 0x50 (Bulk Only Transport)

A Mass Storage device may contain multiple logical units, each represented by a Logical Unit Number (LUN). All logical units on the device share the same device characteristics, but can be addressed independently via their LUN. LUNs are numbered from 0 to 15. If a device does not support multiple LUNs, then 0 is specified for the LUN.

This implementation of the Mass Storage Class provides support for the Bulk Only Transport. In this protocol, three endpoints are utilized:

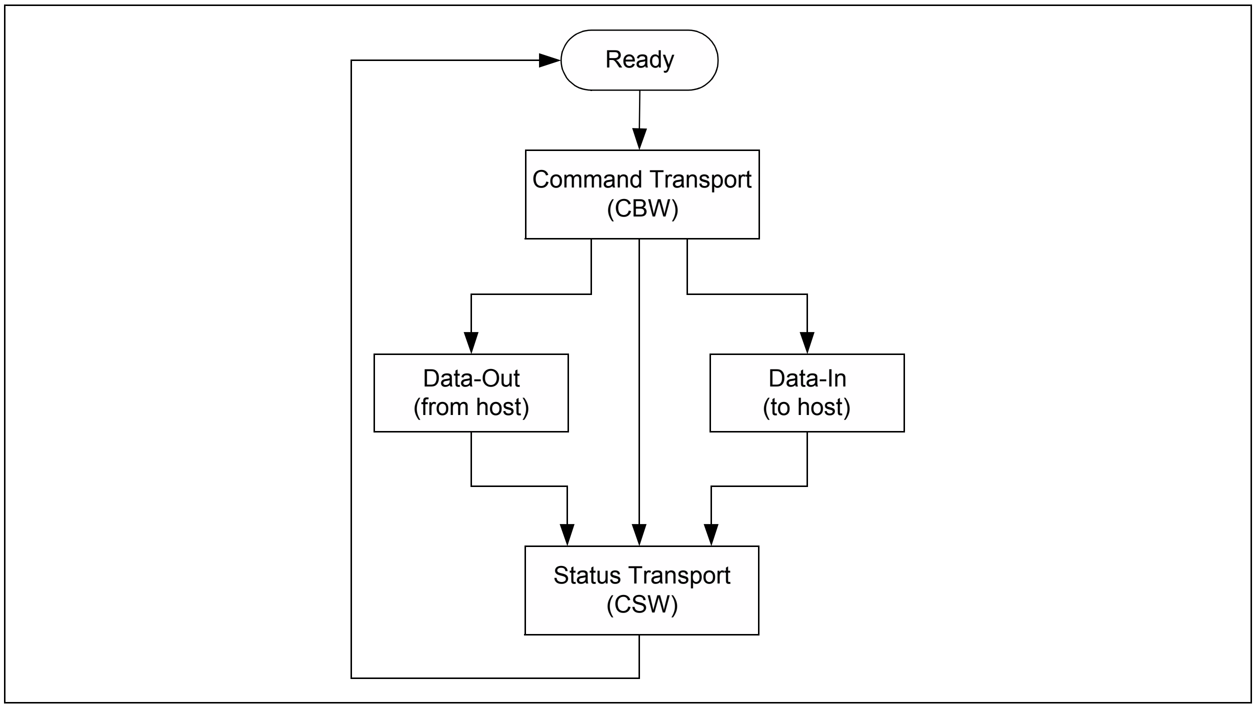
- Endpoint 0 for Control transfers
- One Bulk IN endpoint
- One Bulk OUT endpoint

Bulk transfers consist of three stages:

- Command
- Data (optional)
- Status

The Command stage is sent from the Host to the Peripheral via the Bulk OUT endpoint. The Data stage, if present, utilizes the Bulk IN endpoint if the data is being transferred from the Peripheral to the Host, or the Bulk OUT endpoint if the data is being transferred from the Host to the Peripheral. The Status stage utilizes the Bulk IN endpoint for the Host to receive status information from the Peripheral about the transfer. The flow of the stages is shown in Figure 1.

FIGURE 1: COMMAND/DATA/STATUS FLOW



COMMAND BLOCK WRAPPER (CBW)

The Command Block Wrapper is sent to the Peripheral during the Command phase of the transfer. The CBW is a 31-byte packet that includes the following information:

- Tag to identify the transfer
- Number of bytes to transfer during the Data phase
- LUN to which the transfer applies
- Command block to be executed by the device

The format of the CBW is shown in Table 1.

TABLE 1: COMMAND BLOCK WRAPPER

Command Block Wrapper (CBW)								
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0-3	dCBWSignature							
4-7	dCBWTag							
8-11	dCBWDataTransferLength							
12	bmCBWFlags							
13	Reserved (0)				bCBWLUN			
14	Reserved (0)			BCBMCBLength				
15-30	CBWCB							

The CBW is generated internally by the Mass Storage client driver.

COMMAND STATUS WRAPPER (CSW)

The Command Status Wrapper is sent to the Host from the Peripheral. The CSW is a 13-byte packet that includes the following information:

- Tag to identify the transfer (must match the tag in the CBW)
- The difference between the number of data bytes expected and the number actually transferred
- Success or Failure of the command

The format of the CSW is shown in Table 2.

TABLE 2: COMMAND STATUS WRAPPER

Command Status Wrapper (CSW)								
Byte	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0-3	`dCBWSignature`							
4-7	`dCSWTag`							
8-11	`dCSWDataResidue`							
12	`bCSWStatus`							

The CSW is received and checked internally by the Mass Storage client driver.

USING THE MASS STORAGE CLIENT DRIVER

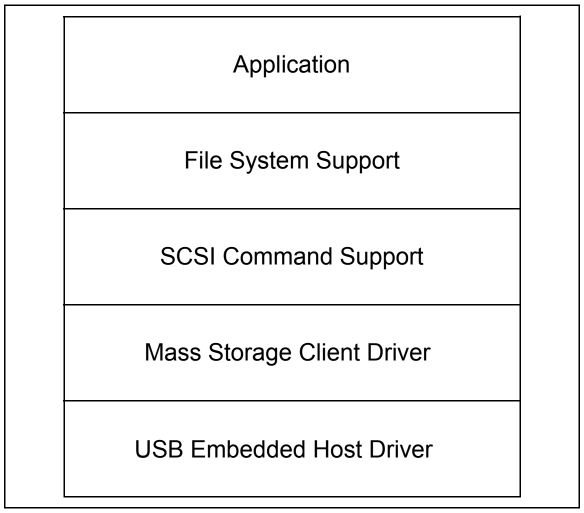
Installing the Mass Storage Client Driver

The Mass Storage Client Driver is installed as a part of complete USB Embedded Host support package, available from the Microchip web site (http://www.microchip.com/usb). Refer to "AN1140 *USB Embedded Host Stack*" for more information on installation.

Application Architecture

Most applications will not interface directly with the USB Host Mass Storage Client Driver. Instead, they will use a media interface layer, which will interface with the Client driver, which in turn will use the host stack driver. For example, the application described by "AN1145 *Using a USB Flash Drive on an Embedded Host*", has five layers including the application layer, as shown in Figure 2.

FIGURE 2: APPLICATION ARCHITECTURE



The "SCSI Command Support" layer is the media interface layer that converts file system commands to SCSI commands and sends them to the USB Peripheral using the USB Mass Storage Class.

Note:	For detailed information about the USB Host Mass Storage Class Driver API, please refer to "AN1141 *USB Embedded Host Stack Programmer's Guide*" and the API documentation provided in the Help directory.

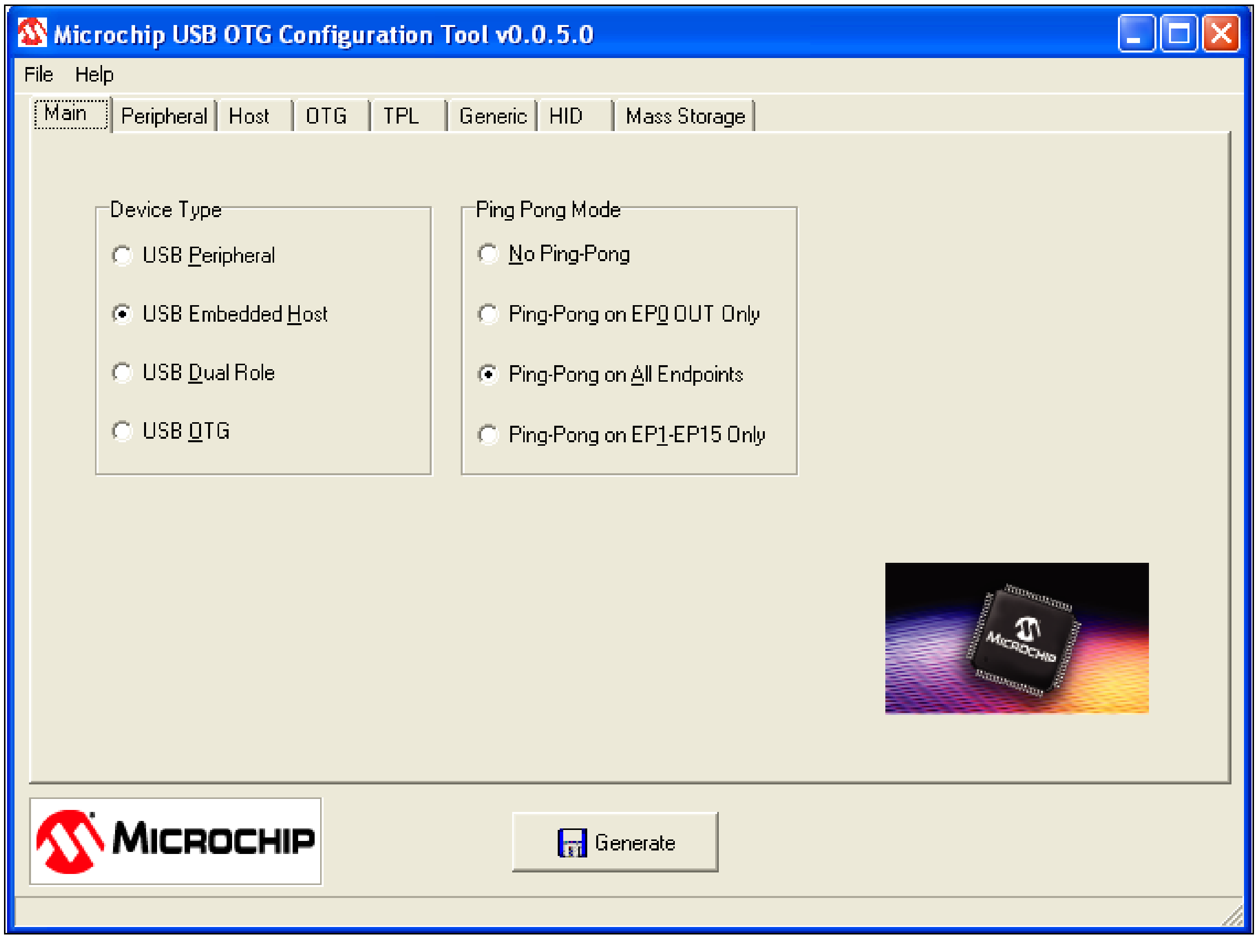
CONFIGURING THE CLASS

Using the USB Configuration Tool

Use the USB configuration tool, `USBConfig.exe`, to configure the Mass Storage client driver for an application. This tool is installed in the `.\Microchip\USB` directory of the installation.

In order to use the Mass Storage client driver for a USB Embedded Host, select the USB device type of the application on the **Main** tab.

FIGURE 3: USB CONFIGURATION - MAIN



Device Type

- USB Peripheral
- [✓ USB Embedded Host]
- USB Dual Role
- USB OTG

Ping Pong Mode

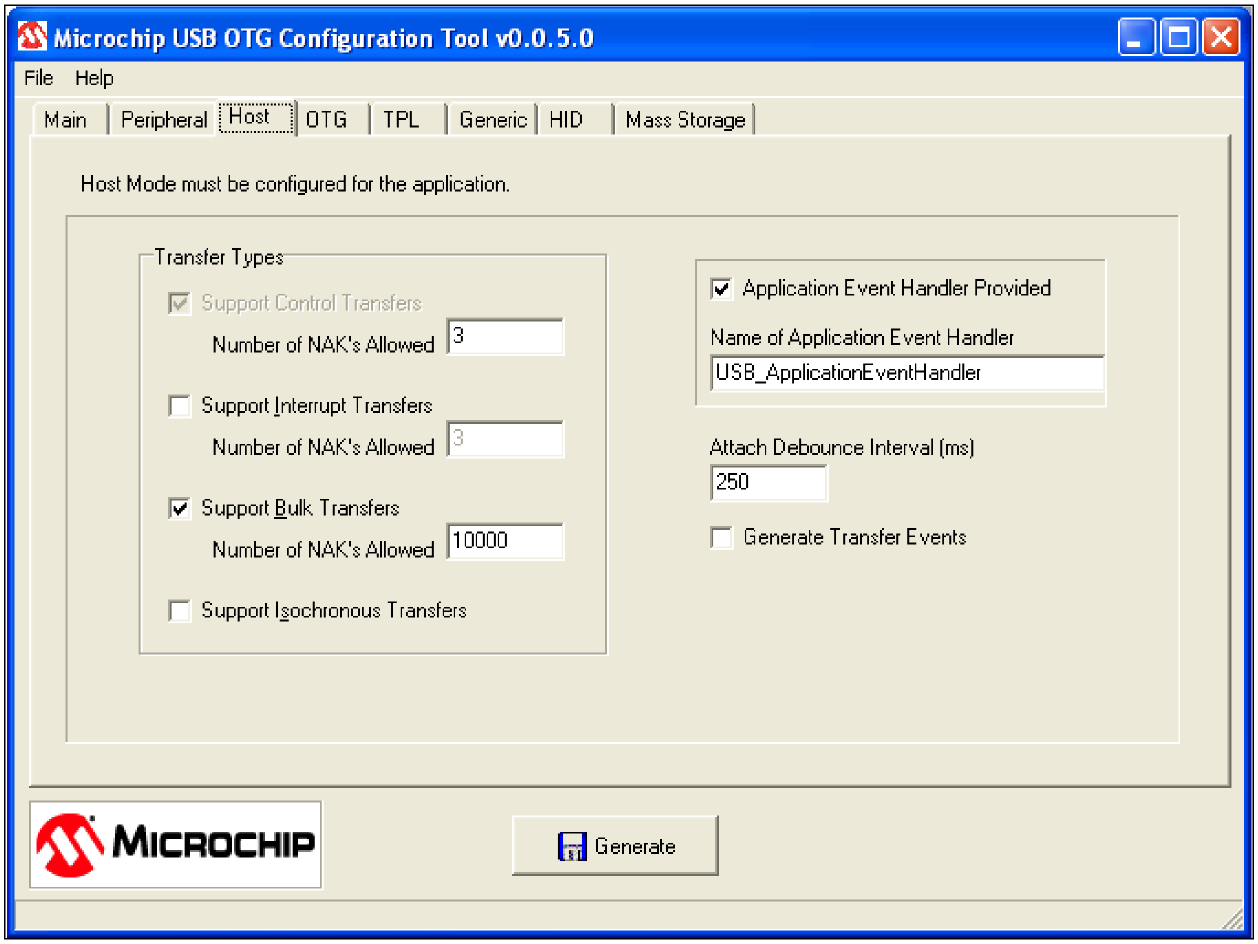
- No Ping-Pong
- Ping-Pong on EP0 OUT Only
- [✓ Ping-Pong on All Endpoints]
- Ping-Pong on EP1-EP15 Only

Select the **Host** tab to configure basic Host operation as shown in Figure 4. The Mass Storage Client Driver requires support for Control and Bulk endpoints. If the application contains no classes that require Interrupt or Isochronous endpoints, then support for those endpoint types can be disabled.

Mass Storage devices can respond rather slowly in comparison to the USB's 1 ms communication frame. Therefore, it is recommended to allow a large number of NAKs before terminating the communication attempt. Also, some devices require longer than the USB specification of 100 ms to initialize after power-up. Therefore, it is recommended to increase the attach debounce interval. Then enter the name of the function in the main source file that serves as the application-level event handler.

The USB Mass Storage Client Driver can either poll the USB Host driver for transfer status or respond to USB Host driver transfer events. Refer to the section **"Event Generation"** below for more information about this selection.

FIGURE 4: USB CONFIGURATION - HOST



Host Mode must be configured for the application.

Transfer Types

- [✓] Support Control Transfers
  - Number of NAK's Allowed: 3
- [ ] Support Interrupt Transfers
  - Number of NAK's Allowed: 3
- [✓] Support Bulk Transfers
  - Number of NAK's Allowed: 10000
- [ ] Support Isochronous Transfers

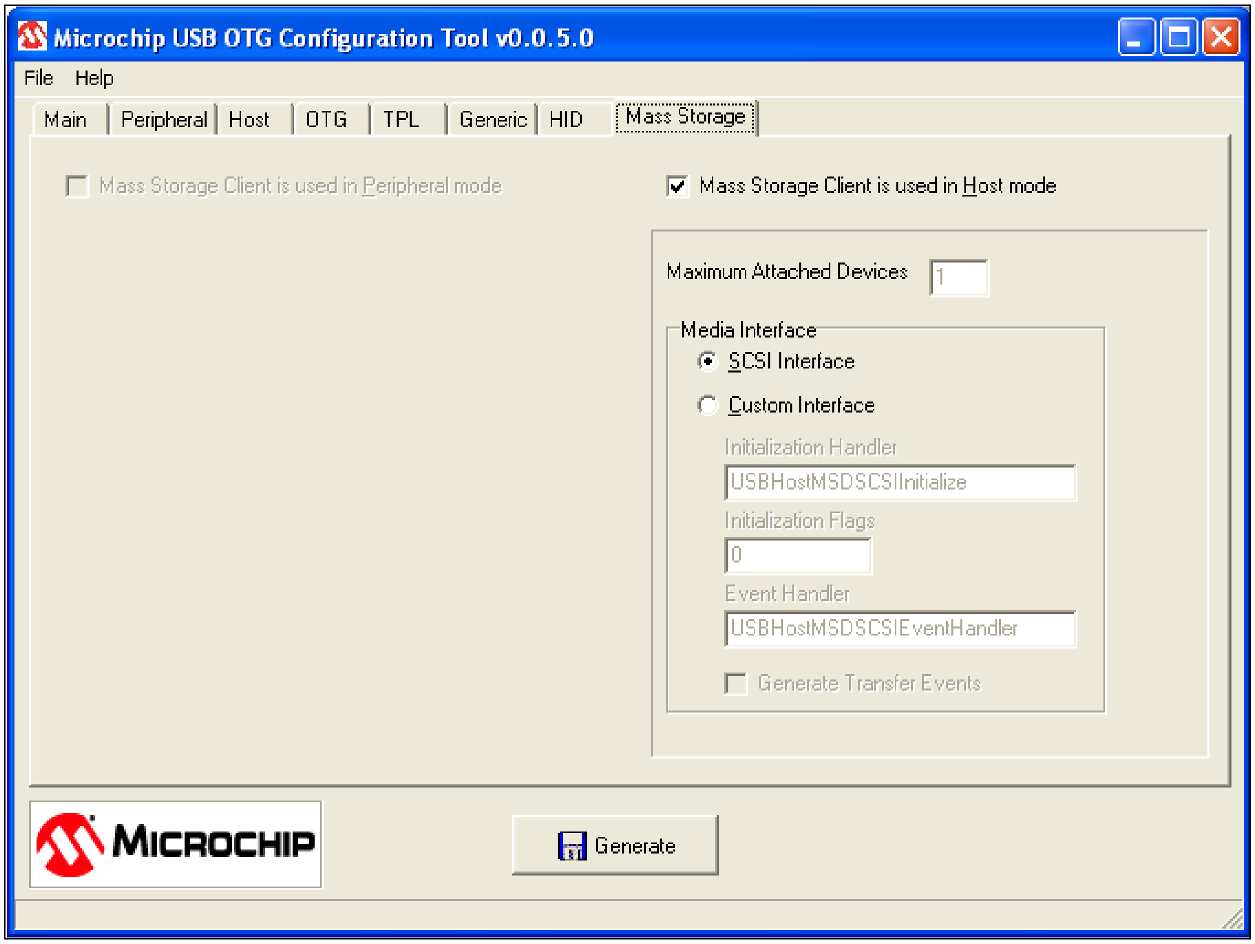
- [✓] Application Event Handler Provided
- Name of Application Event Handler: USB_ApplicationEventHandler
- Attach Debounce Interval (ms): 250
- [ ] Generate Transfer Events

Select the **Mass Storage** tab, check the **Mass Storage Client is used in Host Mode** checkbox to enable support for a Mass Storage Embedded host as shown in Figure 5.

Many Mass Storage devices use a SCSI interface protocol. Support for this protocol is provided with the Mass Storage client driver. Since each function in this layer must complete before the next operation can begin, Mass Storage transfer events are not used.

Click on Generate to create the configuration files, `usb_config.c` and `usb_config.h`, and store them in the project directory.

FIGURE 5: USB CONFIGURATION - MASS STORAGE



Mass Storage Client is used in Peripheral mode [ ]

Mass Storage Client is used in Host mode [✓]

Maximum Attached Devices: 1

Media Interface

[✓ SCSI Interface] Custom Interface

Initialization Handler: USBHostMSDSCSIInitialize

Initialization Flags: 0

Event Handler: USBHostMSDSCSIEventHandler

Generate Transfer Events [ ]

DEFINING THE INTERFACE FUNCTIONS

The client driver requires two interface functions in the media interface layer. The first is the initialization handler, which is called after the Peripheral has been enumerated and initialized by the Mass Storage client driver. The initialization handler should be of the type defined by the `typedef`:

```
typedef BOOL (*USB_CLIENT_INIT) (BYTE address, DWORD flags);
```

This function performs initialization specific to the media interface. If initialization occurs with no error, this routine should return `TRUE`. If errors are encountered, this routine should return `FALSE`, and no transfers to the Peripheral will be allowed.

The second interface function is required to handle events that occur during normal operation. This event handler should be of the type defined by the `typedef`:

```
typedef BOOL (*USB_CLIENT_EVENT_HANDLER) (BYTE address, USB_EVENT event, void *data, DWORD size);
```

For example, one of the events that can occur is EVENT_DETACH. This occurs when a device has detached from the bus. In this case, the media interface layer will need to update its status, by doing operations such as removing the device from its list of attached media.

See the API documentation provided in the Help directory for a complete list of events.

The client driver requires a list of the media interface's required Peripheral initialization and event handlers. This list is defined by the configuration tool `USBConfig.exe`, provided with the stack.

EVENT GENERATION

The client driver can be configured to utilize transfer events (`EVENT_TRANSFER`) from the USB Host layer. In addition, the client driver can be configured to generate transfer events (`EVENT_MSD_TRANSFER`) for the media interface layer. These two events can be configured independently of each other, giving four possible combinations as shown in Table 3 (below).

TABLE 3: EVENT CONFIGURATIONS

USB Host Driver	USB Host MSD Driver
Poll for transfer status	Poll for MSD transfer status
Poll for transfer status	Generate MSD transfer events
Generate transfer events	Poll for MSD transfer status
Generate transfer events	Generate MSD transfer events

If USB Embedded Host transfer events are used, the application will require more program and data memory, but application processing will be performed more efficiently. The USB Embedded Host transfer event configuration is transparent to the media interface layer.

If USB Embedded Host MSD events are used, more program memory is required, and the media interface layer that handles these events must be structured properly. In general, the code architecture required to utilize transfer events is more sophisticated, and more difficult for beginning C programmers to design, develop, debug and maintain.

The choice of whether or not to utilize USB Embedded Host MSD transfer events can also depend on the implementation of the other layers in the application. For example, "AN1045 *Implementing File I/O Functions using Microchip's Memory Disk Drive File System Library"* provides functions to open, close, read from and write to files in a format that PCs can use. Since a user should not be able to write to a file until that file is successfully opened, the implementation of the Memory Disk Drive File System blocks execution of other tasks until the requested operation is complete. Since the File System layer blocks execution, there is no benefit to structuring the media interface layer to utilize USB Embedded Host MSD transfer events. Therefore, the simpler polling mechanism is used.

Note 1: Although the USB Embedded Host utilizes USB interrupts, tranfer event generation from the Host driver layer to the client driver is triggered by a polling mechanism. This is to ensure that the USB ISR completes in a timely fashion. For more information on the host driver , refer to "AN1140 *USB Embedded Host Stack*" and "AN1141 *USB Embedded Host Stack Programmer's Guide*".

2: Regardless of whether or not USB Embedded Host MSD transfer events are used, the media interface layer is required to contain an event handler that processes other system events.

CLIENT DRIVER INITIALIZATION

The Host Mass Storage Client Driver is initialized by a single function:

```
BYTE USBHostMSDInit(void);
```

This function initializes all internal variables for operation. It should only be called once during the application's execution.

The USB Configuration tool will provide a macro `USBInitialize()` to call all of the initialization routines required by the USB Embedded Host driver, the supported client drivers and media interfaces.

NORMAL CLIENT DRIVER OPERATION

Normal background operation is performed by a single function:

```
void USBHostMSDTasks(void);
```

This routine must be called on a regular basis to allow device operation. The polling rate is not critical, since most of the actual transfer of information is handled through the USB interrupt. Since an application may support multiple classes, this function does not call the `USBHostTasks()` function, which also must be called on a regular basis.

The USB Configuration tool will provide a macro `USBTasks()` to call all of the background task routines required by the USB Host driver and the supported client drivers.

SUPPORTED LOGICAL UNIT NUMBERS

If the media interface initialization is successful, the Mass Storage Class driver will immediately inform the media interface layer of the maximum Logical Unit Number of the device via the `EVENT_MSD_MAX_LUN` event. All future transfer requests will be checked against this value to ensure that a valid LUN is being referenced.

Note: The media interface and file system layers may not be able to support multiple LUNs.

PERFORMING A TRANSFER

Communication with a Peripheral is initiated by two functions:

```
BYTE USBHostMSDRead(
   BYTE deviceAddress,
   BYTE deviceLUN,
   BYTE *commandBlock,
   BYTE commandBlockLength,
   BYTE *data,
   DWORD dataLength);
```

```
BYTE USBHostMSDWrite(
   BYTE deviceAddress,
   BYTE deviceLUN,
   BYTE *commandBlock,
   BYTE commandBlockLength,
   BYTE *data,
   DWORD dataLength);
```

The `commandBlock` is a block of up to 16 bytes that tells the Peripheral which operation to perform. When the SCSI media interface layer is used, this block contains the SCSI command to perform the requested operation.

A return code of `USB_SUCCESS (0x00)` indicates that the operation was started successfully.

After initiating communication, take care that `USBHostTasks()` and `USBHostMSDTasks()` are performed while waiting for the operation to complete. The status of the operation can be determined by calling the function:

```
BOOL USBHostMSDTransferIsComplete(
   BYTE deviceAddress,
   BYTE *errorCode,
   DWORD *byteCount)
```

If the function returns `FALSE`, the transfer is not complete, and the returned error code and byte count are not valid. If the function returns `TRUE`, the returned error code indicates the status of the operation, and the returned byte count indicates that how many bytes were transferred.

A transfer of data from the Host to the Mass Storage Peripheral appears as Example 1.

EXAMPLE 1: MASS STORAGE DATA TRANSFER, PERIPHERAL TO HOST

```
error = USBHostMSDRead( device, 0, command, 10, buffer, size );

if (!error)
{
    while (!USBHostMSDTransferIsComplete( device, &error, &count ))
    {
        USBHostTasks();
        USBHostMSDTasks();
    }
}
```

CONCLUSION

The USB Embedded Host Mass Storage class provides a simple interface to popular USB mass storage devices. Embedded applications now can easily take advantage of this flexible, widely available storage media.

RESOURCES

AN1045 “*Implementing File I/O Functions using Microchip’s Memory Disk Drive File System Library*”

- http://www.microchip.com

AN1140 “*USB Embedded Host Stack*”

- http://www.microchip.com

AN1141 “*USB Embedded Host Stack Programmer's Guide*”

- http://www.microchip.com

Universal Serial Bus web site:

- http://www.usb.org

Microchip Technology Inc. web site:

- http://www.microchip.com

WORLDWIDE SALES AND SERVICE

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
http://support.microchip.com
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Kokomo
Kokomo, IN
Tel: 765-864-8360
Fax: 765-864-8387

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario, Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8528-2100
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-4182-8400
Fax: 91-80-4182-8422

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-572-9526
Fax: 886-3-572-6459

Taiwan - Kaohsiung
Tel: 886-7-536-4818
Fax: 886-7-536-4803

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

01/02/08