Universal Serial Bus
Mass Storage Class

Control/Bulk/Interrupt
(CBI) Transport

Revision 1.1
June 23, 2003

Change History

Revision	Issue Date	Comments
1.0	December 14, 1998	Initial release.
1.08	February 6, 2003	CBI_RR020b: Restrict usage of CBI specification to full-speed floppy disk drives only – Redwood City, CA
1.08a	April 3, 2003	CBI_RR021 as amended: clean up references to USB 1.x. Also added new contributors and incorporated bcdUSB editorial modification – Tokyo, Japan.
1.09	May 14, 2003	No objections received. Auto-promote per Tokyo DWG.
1.10	June 23, 2003	No objections received from public comment. Auto-promote per Seattle DWG.

Please send comments via electronic mail to techsup@usb.org

Technical Editors

Curtis E. Stevens	Pacific Digital
Mike Glass	Microsoft Corporation
Mark Williams	Microsoft Corporation
Jim Blackson	Y-E Data, Inc.

Contributors

David G. Lawrence	Global Technology Development
Kenichi Hamada	Y-E Data, Inc.
Toyoko Shimizu	Y-E Data, Inc.
Ryota Okazaki	NEC Corporation
Shigeyoshi Hashi	NEC Corporation
Dominique L'Allement	MGE
Hiromichi Oribe	Hagiwara Sys-Com Co., Ltd.
Yuji Oishi	Hagiwara Sys-Com Co., Ltd.
Jon Eskdale	
Tim Bradshaw	Iomega Corporation
Pat LaVarre	Iomega Corporation
Darrell Redford	Iomega Corporation
Duane Kanz	Microsoft Corporation
Glen Slick	Microsoft Corporation
Jordan Brown	Sun Microsystems, Inc.
Paramita Das	Sun Microsystems, Inc.
Mike Chen	CMD Technology
Calaimany Bhoopathi	Shuttle Technology
Dave Gilbert	In-System Design, Inc.
David Luke	Cypress Semiconductor
Eric Luttman	Cypress Semiconductor
Sadao Yabuki	TEAC System Create Corp
Mike Nguyen	TEAC America, Inc.
Tsuyoshi Osawa	TEAC Corporation
Steven Smith	eTEK Labs
Mike Leibow	eTEK Labs
Yoshitaka Ota	Konica
Albert Saraie	Sicore Systems
Trenton Henry	Standard Microsystems Corporation
Bill Stanley	Adaptec
Shing F. Lin	Adaptec
Alex Afshar	Matsushita Semiconductor
James Quigley	Iomega Corporation
Mike Poulsen	Iomega Corporation
David Jolley	Iomega Corporation
Al Rickey	Phoenix Technologies Ltd.
Mark McCoy	Anchor Chips Inc.
Steve Bayless	Hewlett-Packard

Takashi Matsui	Nanao
Masahiro Ito	Yamagata Fujitsu
Steve Kolokowsky	Cypress Semiconductor
Nathan Obr	Microsoft Corporation
Frits Vanderlinden	Sun Microsystems
Jim Sandman	Iomega Corporation
Bill Russell	Canon
David Cho	Genesys Logic
Sean S. Cho	Genesys Logic
David Sheu	Genesys Logic
Aaron Sun	Genesys Logic
Antonis Lazaridis	TDK Corporation
Hiroki Masui	Standard Microsystems
Kiichi Muto	NEC Electronics

Table of Contents

List of Tables

1. Introduction

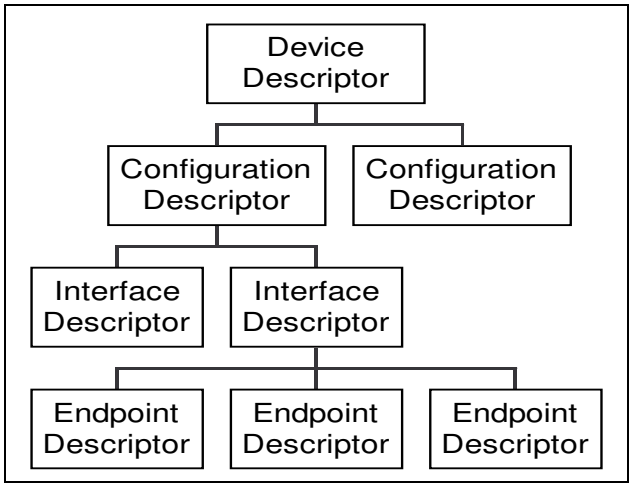
The Universal Serial Bus (USB) is a communications architecture that gives a PC the ability to interconnect a variety of devices via a simple four-wire cable. The USB is actually a two-wire serial communication link that runs at 1.5, 12, or 480 megabits (Mbs) per second. USB protocols can configure devices at startup or when they are plugged in at run time.

How devices on the USB bus behave is the subject of this and other USB Device Class Specifications.

Devices in a given class can be broken into subclasses. These divisions allow software to search the bus and select all of the devices that it can control. Each device can have one or more configurations that define how the device functions. A device that can function in several different ways will have a configuration for each function. A *configuration* is a collection of interfaces. An *interface* specifies which hardware in the device will interact with the USB. Each of these pieces of hardware is called an *endpoint*. Therefore, an interface is a collection of endpoints (for each *alternate setting* of the interface). The USB Device Class Definitions define the default configurations, interfaces, alternate settings, and endpoints that a device in a given class or subclass should provide.

A *descriptor* describes general information about a device, configuration, interface, or endpoint. Figure 1 shows the hierarchical organization of USB descriptors.

Figure 1 - Sample Descriptor Organization



1.1 Scope

- A familiarity with the USB specification is assumed.

This specification addresses the **Control/Bulk/Interrupt (CBI) architecture**, in which the Control, Bulk In/Out and Interrupt endpoints are used for communication between the host and device. That is, the Control endpoint is used to transport command blocks to the device via a class-specific request, as well as to transport standard USB requests. The Bulk In and Bulk Out endpoints are used to transport data between the host and the device. The Interrupt endpoint is used to signal command completion.

This specification assumes the host complies with the USB specification. Nothing in this specification, in its original intent, contradicts the USB specification.

This specification is approved for use only with full-speed floppy disk drives. CBI shall not be used in high-speed capable devices, or in devices other than floppy disk drives. Usage of CBI for any new design is discouraged.

A floppy disk drive shall return in response to an Inquiry (opcode 12h) command, a Peripheral Device Type of 00h and a Removable Media Bit (RMB) set to one which indicates that the media is removable.

1.2 Target Audience

The CBI and Bulk-Only specifications are each intended to be stand-alone documents for the USB Mass Storage sub-class, enabling development of a USB Mass Storage compliant device. A device manufacturer *may* choose to implement both CBI and Bulk-Only, but shall follow each specification as applicable.

The Bootability specification is considered an enhancement to either the CBI or Bulk-Only specifications. Devices *may* be CBI only, Bulk-Only, and not be Bootable. However, to be bootable, the device must comply with the Bootability specification, as well as the CBI specification or the Bulk-Only specification or both.

1.3 Purpose

The purpose of this document is to provide some common descriptions of the USB configuration, interface, and endpoint descriptors, as well as a communications protocol, for operating system, BIOS, and peripheral designers implementing support for CBI mass storage devices. Subclasses are defined that associate device types with industry standard command block definitions. This specification shows how command blocks are delivered to a device and what the device response will be. This provides a framework for designing the peripherals of a given type or subclass.

1.4 Terms and Abbreviations

This section defines important terms used in other sections of this specification. Some of the term definitions are copied from other specifications, such as the *Universal Serial Bus Specification* or the *SCSI Primary Commands – 2 (SPC-2)* specifications.

1.4.1 Terms from the USB Specification

The following term definitions are copied from the *Universal Serial Bus Specification*, Revision 2.0, for the convenience of the reader.

ACK	Handshake packet indicating a positive acknowledgment.
Control Transfer	One of four Universal Serial Bus transfer types. Control transfers support configuration/command/status type communications between client and function.

Endpoint Address	The combination of an endpoint number and an endpoint direction on a USB device. Each endpoint address supports data transfer in one direction.
Endpoint Number	A four-bit value between 0H and FH, inclusive, associated with an endpoint on a USB device.
NAK	Handshake packet indicating a negative acknowledgment.
Stage	One part of the sequence composing a control transfer; i.e., the Setup stage, the Data stage, and the Status stage.

1.4.2 Terms Adapted from the SPC-2 Specification

ASC	An acronym for Additional Sense Code. A device uses ASCs to provide additional detail describing the Sense Key.
ASCQ	An acronym for Additional Sense Code Qualifier. A device uses ASCQs to provide further detail describing the ASC.
Command Block	Also called Command Descriptor Block. A structure defined by the command set used to communicate commands from the host to the device. The command block is carried by an ADSC over the USB during Command Block Transport.
Immediate Command Block	A Command Block which has an immediate bit set, typically starting a long-running operation at the device. The device returns preliminary status information to the host via Status Transport as soon as the command block has been validated.
Sense Data	Data describing an error or exceptional device condition that a device delivers to the host.
SK	An acronym for Sense Key. A device reports generic categories of error and exception conditions in the Sense Key. The host can use the Sense Key information to select high-level error recovery procedures.
Unit Attention Condition	A state that a device maintains while it has asynchronous status information to report to the host. This is not the same as a Persistent Command Block failure.
Unrecoverable Error	A condition in which the device cannot complete the command such as no media, device failure, etc. This is one reason but not the only reason for Command Block failure.

1.4.3 Terms Specific to this Specification

ADSC Acronym for the Accept Device Specific Command class-specific request. The ADSC is the USB wrapper that surrounds a Command Block when transported across the USB.

Class Specific Request All USB devices respond to requests from the host on the device's default pipe. These transfers are made using control transfers. A request is contained in the data portion of a SETUP packet. The *Universal Serial Bus Specification* defines a set of standard requests for all USB devices. When a class of devices is defined, requests specific to that class of devices, which extend the standard requests, can be defined.

Clear stall Use of the USB standard Clear Feature request with the standard ENDPOINT_HALT feature selector. In other words, a Clear Feature (ENDPOINT_HALT) request issued by the host.

Command Not a useful term in this specification. Use the more specific terms Command Block or USB request.

Command Block Failure The condition where a Command Block has completed with a Command Block Status of Failed. This is not the same as Command Transport failure. A stall in Command Block Transport is one way for the device to signal Command Block Failure to the host, but not the only way.

Command Block Status Either In-progress, Passed, or Failed

Command Block Transport In general, Command Block Transport is the way the host sends a Command Block to the device.

Final Status Whether a long running operation begun by an Immediate Command Block is In-progress, or has Passed, or has Failed

May A keyword that indicates an option.

Op Operation code. A two-digit hexadecimal value defined by the command set indicating the command encoded in a Command Block.

Phase Error The result of the device and the host disagreeing on expected data transfer size and/or direction.

Preliminary Status Not the final status, but rather merely the Command Block Status of an Immediate Command Block

Shall A keyword that indicates a requirement.

Should A keyword that indicates flexibility of choice with a strongly preferred alternative; equivalent to the phrase "it is strongly recommended".

Status Transport In general, Status Transport is the way a device sends status to the host.

1.4.4 Key Cross References to USB Core Specifications

This section provides cross-references to key sections of the *USB Specification*, Revision 2.0. The referenced sections of the USB Specification describe rules and concepts that are useful in understanding this specification.

Disallow STALL of Setup Stage of Control Transfer See sections 5.5.5 and 8.5.3.

Port Reset See section 10.2.8.1.

1.5 Bus Trace Notation Conventions

This section outlines the conventions used for gathering bus traces from devices designed using the Control-Bulk-Interrupt USB Mass Storage specification as shown in chapter 2.

Basic conventions used:

- The letter lowercase letter “x” prefixes hexadecimal numbers.
- The “:” (colon) and “ ” (blank) are interchangeable. Colons associate together, visually in a trace, the bytes of a multi-byte field.
- EOP is implied, though not shown, by the trailing “)” parenthesis in the form “DATA(x XX XX …)”
- “DATA( )” is an empty packet (i.e. just EOP from the host Out or from the device In, no other data).
- USB PIDs shown include SOF ACK NAK STALL SETUP OUT IN DATA0 DATA1.
- The bits of a PID can appear in time order of the bus, or reversed.
- ADDR(x7F) is just an example of a device address. x00 to x7F may be valid – the host chooses this value during enumeration.
- ENDP(x1) is just an example of a bulk IN endpoint number. ENDP(x2) is just an example of a bulk OUT endpoint number. ENDP(x3) is just an example of an interrupt endpoint number. x1 to x1F may be valid – the device reports this value during enumeration. Because the endpoint number and the direction together form a distinct endpoint address, the endpoint number may be the same for bulk IN and bulk OUT, or for interrupt IN and bulk OUT. ENDP(x0) is the required default control endpoint number.
- The choice of DATA0 and DATA1 are examples. Bulk and interrupt traffic in particular often begin with one or the other. The *USB Specification* gives rules for when the host changes its choice of this “data toggle” and how the device responds to the two different choices at different times.
- NAK in examples shows the possibility of the NAK, without requiring that NAK.
- Repeat of NAK is not shown.
- SOF in examples shows the possibility of the SOF, without requiring that SOF.
- IDLE is not shown.
- Bit-stuffing, CRCs, and idle times are not shown.
- No babble shown, no violation of *MaxPacketSize* of endpoint shown.

2. Functional Characteristics

The host and mass storage devices communicate over the USB using the protocols described by this specification. These protocols are based on the command/data/status style of communication. That is, the protocols transport command blocks from the host to the device; transport data between the host and device; transport "status" of the command block from the device to the host; and allow the host to reset the device.

Mass storage devices shall accept two types of resets and three types of command blocks.

The two types of resets are:

- Port Reset
- Command Block Reset

The three types of command blocks are:

- Non-Data
- Data-In
- Data-Out

The three types of command blocks, with their respective protocols, including status and error reporting, are covered below. The host decides which protocol to try. The device then decides whether to cooperate. The "Status" of a command block is given by the device and is one of In-Progress, Passed, or Failed.

This specification does not define how the device responds if the host fails to comply with the requirements of this specification (often expressed using the phrase "the host shall …"). A host that discovers it has violated protocol inadvertently should reset the device.

2.1 Port Reset Protocol

See the *USB Specification*.

If a mass storage device does not respond as outlined in this specification, the host may resort to a port reset of the offending device.

WARNING: Using Port Reset to abort an In-Progress command may result in the loss of data.

2.2 Command Block Reset Protocol

Before taking the drastic action of a Port Reset, the host may first try a Command Block Reset.

To issue a Command Block Reset, the host shall use the Non-Data Command Protocol to transport the command block:

```
1Dh 04h FFh FFh FFh FFh ...
```

The device may use the trailing FFh bytes to distinguish the Command Block Reset from the legacy op 1Dh SEND DIAGNOSTIC command block.

Upon receipt of a Command Block Reset, the device shall

1. Attempt to abort any currently executing command or data transfer;
2. Drop any command blocks that may be queued for execution;
3. Discard any data buffered at the device;
4. Reset the mass storage portion of the device, as described in the corresponding industry standard documents indicated by the *bInterfaceSubClass* field (i.e. perform a self-test, initialize the device's operating modes to there default conditions);
5. Clear any persistent command block failure condition.

After a Command Block Reset completes, the Stall condition and data toggle of the device's endpoints are undefined. The host shall send a Clear Feature (ENDPOINT_HALT) USB standard request to both the Bulk In and Bulk Out pipes to clear any Stall conditions and reset the data toggles.

WARNING: Using Command Block Reset to abort an In-Progress command may result in the loss of data.

2.3 Non-Data Command Protocol

The host shall use the Non-Data Command Protocol to transport a command block and its status with no accompanying data.

2.3.1 Command Transport for Non-Data Commands

The host shall send each command block to the device via a Control Transfer to the default control pipe (endpoint number zero). The host shall begin the Command Transport with a class-specific request (the ADSC). The host shall send the Command Transport Data in the Data Stage of the Control Transfer. The Command Transport is completed with the completion of the Status Stage of the Control Transfer.

The format, length, and padding of the command block is defined by the command set of the device, as indicated by the Subclass code (see the *USB Mass Storage Class Specification Overview*.).

According to the *USB Specification*, the host shall fragment the command block and toggle between DATA0 and DATA1 as needed to respect the *MaxPacketSize* reported in the Device descriptor.

The device may delay Command Transport indefinitely by NAK in the Data Stage and/or Status Stage of the Control Transfer, subject to the limitations expressed in the rest of this specification.

Note: The previous paragraph states a class-specific exemption to the *USB Specification*.

The device may fail the Command Transport by STALL in the Data Stage or Status Stage of the Control Transfer. Else the device shall accept the Command Transport Data by ACK in the Data Stage and Status Stage of the Control Transfer.

Successful transport of Command Transport Data does not necessarily imply the status of the command block contained therein is Passed. See the discussion of Status Transport.

Table 2.1 shows an example of the ADSC class-specific request.

Table 2.1 – Example of an ADSC Class-Specific Request

```
1   SOF(xA5)
2   SETUP(xB4) ADDR(x7F) ENDP(x0)
3       DATA0(xC3) DATA(x 21 00 00:00 00:00 0C:00)
4       ACK(x4B)
5   SOF(xA5)
6   OUT(x87) ADDR(x7F) ENDP(x0)
7       DATA1(xD2) DATA(x 2A 00 00:01:23:45 00 00:)
8       NAK(x5A)
9   SOF(xA5)
10  OUT(x87) ADDR(x7F) ENDP(x0)
11      DATA1(xD2) DATA(x 2A 00 00:01:23:45 00 00:)
12      ACK(x4B)
13  SOF(xA5)
14      DATA0(xC3) DATA(x 00 00 00:00)
15      NAK(x5A)
16  SOF(xA5)
17  OUT(x87) ADDR(x7F) ENDP(x0)
18      DATA0(xC3) DATA(x 00 00 00:00)
19      ACK(x4B)
20  SOF(xA5)
21  IN(x96) ADDR(x7F) ENDP(x0)
22      NAK(x5A)
23  SOF(xA5)
24  IN(x96) ADDR(x7F) ENDP(x0)
25      DATA1(xD2) DATA(   )
26      ACK(x4B)
```

USB requires the device to ACK the Setup Stage. The device may STALL in place of the ACKs of the Data Stage or Status Stage to indicate command block failure.

2.3.2 Status Transport for Non-Data Commands

To indicate a command block is In-progress, the device shall NAK the appropriate endpoint, as described below. To indicate a command block has passed, the device shall ACK the appropriate endpoint and may supply status bytes by interrupt pipe, as specified in section 2.3.2.2. To indicate a command block has failed, the device shall STALL a pipe or supply status bytes by interrupt pipe, or both. Details follow.

2.3.2.1 Status Transport by Control Pipe

The device may choose to transport command block status by control pipe.

The device may NAK the Data Stage and/or the Status Stage of the ADSC class-specific request to indicate a command block is In-Progress. But an ACK of the Status Stage does not mean command block complete; that ACK only means the command block transport has completed.

The device may STALL the Data Stage and/or the Status Stage of the ADSC class-specific request to indicate a command block has Failed. If the device does STALL an ADSC, then the device shall not transport the status for the command block transported by the stalled ADSC by interrupt pipe – thus the host shall not wait for command block status to appear there.

Table 2.1 of Command Transport shows an example of the transport of In-Progress command block status by control pipe. The device may STALL in place of any of the NAKs of that example to transport Failed command block status.

2.3.2.2 Status Transport by Interrupt Pipe

This section applies to devices that implement command completion interrupts.

After successful completion of Command Transport, the device may NAK the interrupt pipe (in response to IN PID) to indicate a command block is In-Progress.

If and when the command block status changes from In-progress to Passed or Failed, the device shall then return status bytes on the Interrupt endpoint. The status bytes returned depend on the command set, as described in Chapter 3.

The command set defines whether or not any bulk data transported for a command block is valid if the command block fails. When a command block fails, the bulk data transported prior to the failure may or may not be valid, as defined in the command set.

Table 2.2 shows an example of the transport of command block status by interrupt pipe.

Table 2.2 – Example of Command Block Status Transport by Interrupt Pipe

```
1    SOF(xA5)
2    IN(x96) ADDR(x7F) ENDP(x3)
3      NAK(x5A)
4    SOF(xA5)
5    IN(x96) ADDR(x7F) ENDP(x3)
6      DATA1(xD2) DATA(xFF:FF)
7      ACK(x4B)
```

2.3.2.3 Host Violation of Status Transport Protocol

If the host does not poll the Interrupt endpoint and proceeds to send another ADSC, the device shall clear the still pending interrupt when the device accepts the ADSC.

2.3.2.4 Indefinite Delay

The device may NAK indefinitely, for example, when the device interprets a command block as a request to move data, but the host does not move data. In this case, the host shall time out the offending request and reset the device as described by this specification. How long the host waits before timing out is command set dependent.

Table 2.1 of Command Transport and Table 2.2 of Status Transport by Interrupt Pipe show where NAK may appear, thus where indefinite NAK may appear.

2.3.2.5 Status Transport for Immediate Command Blocks

Some command sets define both non-immediate and immediate command blocks. Immediate command blocks transport “preliminary status” as soon as the device is ready to interpret another command block.

The Immediate command block may begin a long-running operation in the device. The device shall transport the preliminary status using the Status Transport defined herein. The command set defines how the final status of the Immediate command is transported.

2.3.2.6 Persistent Command Block Failure

After the device fails a command block, the device may continue to fail every command block received until the host either supplies a command block that clears this Persistent Command Block Failure, performs a Command Block Reset, performs a USB Suspend, or performs a Port Reset.

The command set defines

- Which errors persist and which do not;
- Which command blocks, if any, clear a Persistent Command Block Failure;
- Whether such a command block proceeds to execute as normal;
- Which command blocks may complete successfully without disturbing an otherwise Persistent Command Block Failure.

While making a command block failure persist, the device may transport the failed status using any of the normal mechanisms. That is, the device may ACK the transport of a command block and then fail the command block later.

While making a command block failure persist, the device may change the ASC/ASCQ values transported by interrupt pipe.

Because a USB Suspend may deny power to a bus-powered device, the device may forget a Persistent Command Block Failure in response to a USB Suspend.

2.4 Data In Command Protocol

The host shall use the Data-In Command Protocol to transport a command block, read data in from a device, and transport the command block status.

2.4.1 Command Transport for Data In Commands

See section 2.3.1 Command Transport for Non-Data Commands.

2.4.2 Data In Transport

2.4.2.1 Data In: Normal Operation

The host shall send the IN PID to the Bulk In endpoint to request transport of data in from the device. The host shall continue to send IN PID until it receives the amount of data expected.

The device shall NAK until it has data available. When data becomes available, the device shall supply data in *MaxPacketSize* packets, as described in the USB specification. If the supply of data is temporarily exhausted, the device shall again NAK until data becomes available.

If the device supplies a zero-length or short packet, the host should consider the transport of data for the current command block to have ended, and send no more IN PID.

The device shall send a zero-length or short packet as the last packet only if the total amount of data supplied is less than that expected by the host. The device shall send a short, but not a zero-length, last packet if the amount of data remaining does not completely fill a Bulk In packet. The device shall not send a zero-length packet if the total amount of data supplied exactly matches that expected by the host and is a multiple of *MaxPacketSize*.

2.4.2.2 Data In: Short Transfer, Normal Operation

The command set may define legitimate cases where the device supplies less data than requested by the host, for example, op 5Ah MODE SENSE command. In this condition, the device shall send a zero-length or short packet to identify the end of the data.

If the device supplies a zero-length or short packet, the host should consider the transport of data for the current command block to have ended, and send no more IN PID.

2.4.2.3 Data In: Host Requests Too Few Bytes

If the host sends less IN PID than the device expects, the device cannot know if the host has stopped transporting data or if the host is just remarkably slow. Therefore, in this case, the device may NAK the transport of status indefinitely.

2.4.2.4 Data In: Data Overrun within Packet

If the device sends more data in one packet than the host expects, the host may reset the device as described by this specification.

2.4.2.5 Data In: Host Requests Too Many Bytes

If the host sends more IN PID than the device can satisfy, then the device may NAK indefinitely (for more information, see section 2.3.2.4). The device should STALL (see section 2.4.3).

2.4.2.6 Data In: Perhaps the Wrong Bytes

The device may detect an error after all data has been successfully transported, in which case the device shall indicate the error in the Status Transport. The host shall read the Status Transport to determine the validity of the data.

2.4.3 Status Transport for Data In Commands

See section 2.3.2 Status Transport for Non-Data Commands.

2.4.3.1 Status Transport by Bulk Pipe

2.4.3.1.1 Do Not Report Stalled ADSC

If the device reports the command block status by STALL of the ADSC class-specific request, the device shall not report that status again by interrupt pipe nor by bulk pipe. That is, the device shall not make command completion status available by interrupt and the device shall not stall either bulk endpoint.

2.4.3.1.2 Status by Bulk In Pipe

The device may choose to report status by Bulk In pipe in addition to reporting status by Status Transport as described by this specification (see section 2.3.2).

The device may NAK the Bulk In endpoint (in response to an IN PID respectively) to indicate a command block is In-Progress.

The device may STALL the Bulk In pipe to indicate a command block has failed.

2.4.3.1.3 Clearing a Bulk In Pipe Stall

According to the *USB Specification*, the host shall respond to a STALL of the Bulk In pipe by the device with a Clear Feature of Endpoint Halt to clear the Stall and reset the data toggle to DATA0.

A device may STALL the Bulk In pipe AFTER the last data has transferred. For this reason the host shall also issue Clear Feature for Endpoint Halt to the Bulk In pipe if the device reports that the Data In command block has Failed.

2.5 Data Out Command Protocol

The host shall use the Data-Out Command Protocol to transport a command block, write data out to the device, and transport the command block status.

2.5.1 Command Transport for Data Out Commands

See section 2.3.1 Command Transport for Non-Data Commands.

2.5.2 Data Out Transport

2.5.2.1 Data Out: Normal Operation

The host shall send the OUT PID to the Bulk Out endpoint to request to transport data out to the device.

The device may ACK to accept data. If the host supplies a short packet, the device should consider the transport of data for the current command block to have ended.

The device shall NAK until it has buffer available to accept data.

2.5.2.2 Data Out: Host Sends Too Few Bytes

If the host sends less OUT PID than the device expects, the device cannot know if the host has stopped transporting data or if the host is just slow. Therefore, in this case, the device may NAK the transport of status indefinitely.

2.5.2.3 Data Out: Host Sends Too Many Bytes

If the host sends more OUT PID than the device can accept, then the device may NAK indefinitely, STALL the Bulk Out pipe, or ACK the data in an implementation-specific manner.

2.5.3 Status Transport for Data Out Commands

See section 2.4.3, except Bulk In reads as Bulk Out.

2.6 Unidirectional Data Transport Requirement

The command sets applied to this specification shall not define a command block that requires data to be transferred on both the Bulk In and Bulk Out pipes.

3. Standard Descriptors

CBI mass storage devices shall support the following standard USB descriptors:

- **Device**. Each device has one device descriptor.
- **Configuration**. Each device has one default configuration descriptor, which supports at least one interface.
- **Interface**. CBI mass storage devices shall support at least a Data interface. Some devices may support additional interfaces based on their control and isochronous capabilities.
- **Endpoint**. CBI mass storage devices shall support the following endpoints, in addition to the Endpoint 0 for control transfers that is required of all USB devices:
  - Bulk In endpoint. CPU data transfers.
  - Bulk Out endpoint. CPU data transfers.
  - Interrupt endpoint. Command completion interrupt, required if *bInterfaceProtocol* is 00h.

CBI mass storage devices have no class-specific descriptors.

The rest of this section describes the standard USB device, configuration, interface, and endpoint descriptors for CBI mass storage devices. For information about other standard descriptors, see Chapter 9, "USB Device Framework," of the *USB Specification*.

3.1 Device Descriptor

There is only one Device Descriptor for each USB device. Note that for a Mass Storage class device, the device class and subclass codes are not specified in the Device descriptor, but in the Interface descriptor (for more information, see section 3.3).

Table 3.1 - Device Descriptor

Offset	Field	Size	Value	Description
0	bLength	Byte	12h	Size of this descriptor in bytes.
1	bDescriptorType	Byte	01h	DEVICE descriptor type.
2	bcdUSB	Word	????h	USB Specification Release Number in BCD. At the time this specification was published, the recommended value was 0200h.
4	bDeviceClass	Byte	00h	Device class is indicated in the interface descriptors.
5	bDeviceSubClass	Byte	00h	Device subclass is indicated in the interface descriptors.
6	bDeviceProtocol	Byte	00h	Mass Storage devices do not use class-specific protocols.

Offset	Field	Size	Value	Description
7	bMaxPacketSize0	Byte	??h	Maximum data transfer size can be 8, 16, 32 or 64 bytes.
8	idVendor	Word	????h	Vendor ID (assigned by the USB).
10	idProduct	Word	????h	Product ID (assigned by the manufacturer).
12	bcdDevice	Word	????h	Device release number in BCD.
14	iManufacturer	Byte	??h	Index of string descriptor describing manufacturer.
15	iProduct	Byte	??h	Index of string descriptor describing this product.
16	iSerialNumber	Byte	??h	Index of string descriptor describing the device's serial number. A device may provide a unique serial number.
17	bNumConfigurations	Byte	??h	Number of possible configurations. There must be at least one default configuration.

If the device does not have a serial number, *iSerialNumber* shall be set to 00h. If the device does have a serial number, *iSerialNumber* shall be set to the index of the string descriptor that contains the serial number.

If provided, the serial number must be unique to each USB Vendor ID and Product ID pair. The serial number shall contain 12 hexadecimal digits, represented as a 12-character Unicode string. Each character in the Unicode string shall be uppercase alphanumeric and printable.

The optional serial number allows a globally unique identifier (GUID) to be generated by concatenating the 16-bit vendor ID, *idVendor,* and the 16-bit product ID, *idProduct,* and the value represented by the string descriptor indexed by *iSerialNumber*.

Note that the field *iSerialNumber* is an index to a String Descriptor and does not contain the string itself.

3.2 Configuration Descriptor

A Mass Storage device has one default configuration descriptor. This descriptor has at least one interface. The *bLength* field of the configuration descriptor specifies the length of this descriptor, not the length of the data returned by the Get Configuration request.

Table 3.2 - Configuration Descriptor

Offset	Field	Size	Value	Description
0	bLength	Byte	09h	Size of this descriptor in bytes.
1	bDescriptorType	Byte	02h	CONFIGURATION descriptor type.

Offset	Field	Size	Value	Description
2	wTotalLength	Word	????h	Number of bytes in this configuration. This includes the configuration descriptor plus all of the interface and endpoint descriptors.
4	bNumInterfaces	Byte	??h	Mass storage devices must support at least a data interface.
5	bConfigurationValue	Byte	??h	Value to use as an argument to the SetConfiguration() request to select this configuration.
6	iConfiguration	Byte	??h	Index of string descriptor describing this configuration.
7	bmAttributes	Byte	?0h	Configuration characteristics: Bit Description 7 Reserved, set to 1 6 Self-powered 5 Remote wakeup 4-0 Reserved, set to 0 For a full description of this bmAttributes bitmap, see the *USB Specification*.
8	MaxPower	Byte	??h	Maximum power consumption of this configuration. Expressed in 2mA units (i.e., 50 = 100mA).

3.3 Interface Descriptors

All Mass Storage devices shall support at least one interface, the Data Interface. The Data Interface includes up to four endpoints, the default Control endpoint, the Bulk In and Bulk Out endpoints that allow data to be transferred to or from the device, and the optional Interrupt endpoint for signaling command completion.

Composite mass storage devices may support additional interfaces, for example control, audio, or video capabilities. These interfaces are described in the USB device class specification appropriate to the function. When the default configuration is selected, only the Data interface is enabled. This allows minimal control of mass storage drives by software with relatively limited capabilities, such as a system BIOS.

Mass storage devices may have multiple alternate settings. To boot, the host should examine each of the alternate settings to look for a *bInterfaceProtocol* it supports optimally.

Table 3.3 - Data Interface Descriptor

Offset	Field	Size	Value	Description
0	bLength	Byte	09h	Size of this descriptor in bytes.
1	bDescriptorType	Byte	04h	INTERFACE descriptor type.
2	bInterfaceNumber	Byte	0?h	Zero-based value identifying the number of this interface.

Offset	Field	Size	Value	Description
3	bAlternateSetting	Byte	??h	Value used to select an alternate interface.
4	bNumEndpoints	Byte	0?h	Number of endpoints used by this descriptor. The value of this field must be 02h if the optional Interrupt endpoint is not used in this interface or 03h if the optional Interrupt endpoint is used in addition to the two Bulk endpoints.
5	bInterfaceClass	Byte	08h	This interface is specific to Mass Storage devices.
6	bInterfaceSubClass	Byte	04h	UFI Subclass codes indicate which industry standard command block definition to use. For a list of the valid subclass codes, see the *USB Mass Storage Class Specification Overview*.
7	bInterfaceProtocol	Byte	00h or 01h	Indicates the command protocol used by the interface: See the *USB Mass Storage Class Specification Overview* for a description of protocol 00h and 01h.
8	iInterface	Byte	??h	Index to string describing this interface.

Note that the Subclass code values used in the *bInterfaceSubClass* field specify the industry-standard specification that describes the command block definition used by the interface; these Subclass codes do not specify a type of storage device (such as a CD-ROM or floppy disk drive). For more information, see the *USB Mass Storage Class Specification Overview*.

The value in the *bInterfaceProtocol* field specifies the kind of protocol used by the interface. See the *USB Mass Storage Class Specification Overview*.

3.4 Endpoint Descriptors

A Mass Storage device shall support a minimum of three endpoints: control, bulk in, and bulk out.

A CBI Mass Storage device may also support an Interrupt endpoint.

Every USB device also defines a Control endpoint (Endpoint 0). This is the default endpoint and does not require a descriptor. The Device descriptor specifies the maximum packet size of the Control endpoint. The Control endpoint is used for both global USB commands and device-specific commands.

3.4.1 Bulk In Endpoint

The Data-In Command Protocol uses the Bulk In endpoint to transfer data from the device or to return status information about the device.

Table 3.4 - Bulk In Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	Byte	07h	Size of this descriptor in bytes.
1	bDescriptorType	Byte	05h	ENDPOINT descriptor type.
2	bEndpointAddress	Byte	8?h	The address of this endpoint on the USB device. This address is an endpoint number between 1 and 15. Bit 0..3 Endpoint number Bit 4..6 Reserved, must be 0 Bit 7 0 = Out, 1 = In
3	bmAttributes	Byte	02h	This is a Bulk endpoint.
4	wMaxPacketSize	Word	00??h	Maximum data transfer size can be 8, 16, 32 or 64 bytes.
6	bInterval	Byte	00h	Does not apply to Bulk endpoints.

3.4.2 Bulk Out Endpoint

The Data-Out Command Protocol uses the Bulk Out endpoint to transfer data from the host to the device. This may be data to be stored on the device media or additional command parameter information.

Table 3.5 - Bulk Out Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	Byte	07h	Size of this descriptor in bytes.
1	bDescriptorType	Byte	05h	ENDPOINT descriptor type.
2	bEndpointAddress	Byte	0?h	The address of this endpoint on the USB device. This address is an endpoint number between 1 and 15. Bit 0..3 Endpoint number Bit 4..6 Reserved, must be 0 Bit 7 0 = Out, 1 = In
3	bmAttributes	Byte	02h	This is a Bulk endpoint.
4	wMaxPacketSize	Word	00??h	Maximum data transfer size can be 8, 16, 32 or 64 bytes.
6	bInterval	Byte	00h	Does not apply to Bulk endpoints.

3.4.3 Interrupt Endpoint

If *bInterfaceProtocol* is 00h, the CBI mass storage device shall support an interrupt endpoint, and use it to signal command completion as described in section 2.3.2.2. The CBI device shall generate this interrupt no more frequently than once per command block. The data block sent by a command completion interrupt is described below.

If *bInterfaceProtocol* is 01h, the CBI device may choose to implement an Interrupt endpoint. If implemented, the use of such an Interrupt endpoint is not defined by this specification.

Table 3.6 - Interrupt Endpoint Descriptor

Offset	Field	Size	Value	Description
0	bLength	Byte	07h	Size of this descriptor in bytes.
1	bDescriptorType	Byte	05h	ENDPOINT descriptor type.
2	bEndpointAddress	Byte	8?h	The address of this endpoint on the USB device. This address is an Endpoint number between 1 and 15. Bit 0..3 - Endpoint number Bit 4..6 - Reserved, must be 0 Bit 7 - 0 = Out, 1 = In
3	bmAttributes	Byte	03h	This is an Interrupt endpoint.
4	wMaxPacketSize	Word	0002h	This returns 2 bytes of status data.
6	bInterval	Byte	??h	The rate at which the interrupt endpoint may send data. For example, if the value is FFh, then the interrupt endpoint may send data once every 255 frames.

3.4.3.1 Interrupt Data Blocks

To generate an interrupt, the CBI device shall return two bytes of status information on the Interrupt endpoint. The host system software shall use the value of the *bInterfaceSubClass* field in the Interface descriptor to interpret the meaning of the two bytes. The valid values for the *bInterfaceSubClass* are specified in the *USB Mass Storage Class Specification Overview*..

3.4.3.1.1 Common Interrupt Data Block

Unless specified differently below, all devices shall return an interrupt data block as defined in this section (if the device supports an interrupt).

The first byte the device returns contains an interrupt type code and the second byte contains a value associated with that interrupt type.

Table 3.7 – Interrupt Data Block

Offset	Field	Size	Value	Description
0	bType	Byte	??h	00h: Command completion interrupt 01h - FFh: Reserved
1	bValue	Byte	??h	If bType = 00h, then Bit 7…4: Vendor-specific Bit 3…2: Reserved (shall be zero and shall be ignored by the host system software) Bit 1…0: Command block status 00 = Pass 01 = Fail 10 = Phase Error 11 = Persistent Failure

In response to a persistent failure, the host shall send an op 03h REQUEST SENSE command block.

In response to a phase error, the host shall attempt to reset the device using a Command Block Reset. As a last resort, the host may reset the device using a USB port reset. Note that a USB port reset will reset all interfaces of a multiple-interface device, not only the mass storage Data interface.

The behavior of a device in response to a phase error is undefined by this specification.

3.4.3.1.2 Special Operations for Interrupt Data Block for bInterfaceSubClass = 01h

RBC devices shall support the op 03h REQUEST SENSE command as specified in the *SCSI Primary Commands – 2* (SPC-2) specification. The sense codes returned by the REQUEST SENSE command are the status codes documented in the RBC specification as well as the request sense codes documented in the SPC-2 specification.

3.4.3.1.3 Interrupt Data Block for bInterfaceSubClass = 04h

The first byte the device returns contains a device-specific ASC code and the second byte contains a device-specific ASCQ code. For the valid values of the *bASC* and *bASCQ* fields, see *UFI Command Specification*.

Table 3.8 – Interrupt Data Block for bInterfaceSubClass = 04h

Offset	Field	Size	Value	Description
0	bASC	Byte	??h	ASC: Additional Sense Code
1	bASCQ	Byte	??h	ASCQ: Additional Sense Code Qualifier

In response to a Persistent Command Block Failure, the host shall send an op 03h REQUEST SENSE command block.

4. Requests

A Mass Storage device can respond to two different types of requests:

- Standard USB device requests, which perform general functions for supporting the bus and bus-related functions.
- Class specific requests, which cause the device to transfer data to or from the device.

A Mass Storage device shall support all of the required standard device requests described in Chapter 9 of the *USB Specification*.

4.1 Class-Specific Requests

A CBI Mass Storage device shall support one USB class-specific request on the Control endpoint (Endpoint 0):

- **Accept Device-Specific Command** (ADSC)

The Accept Device-Specific Command class-specific request is used by the CBI Command Transport Protocols to send a command block from a host to a device. The ADSC setup packet has eight bytes as follows:

Table 4.1 - Accept Device-Specific Command

Byte	Field	Size	Value	Description
0	bmRequestType	Byte	21h	Characteristics of request: class specific to device
1	bRequest	Byte	00h	Device-Specific request code
2	(LSB) wValue (MSB)	Word	00h	reserved: zero
3			00h	
4	(LSB) wIndex (MSB)	Word	??h	bInterfaceNumber
5			00h	reserved: zero
6	(LSB) wLength (MSB)	Word	????h	Command Transport Data size in bytes
7				

bmRequestType denotes this request as a class specific request from the host to the interface.

Table 4.2 - ADSC bmRequestType

	D7	D6	D5	D4	D3	D2	D1	D0
Desc	Direction	Type		Recipient				
Value	0	0	1	0	0	0	0	1

D7	**Data Transfer Direction:**	0 = Host to device.
D6-5	**Request Type:**	1 = Class-specific request
D4-0	**Recipient:**	1 = Interface

bRequest indicates the specific request. A value of 0 denotes an ADSC request. All other values are reserved.

wValue is reserved.

wIndex, bits 7-0: the host shall set this field to *bInterfaceNumber*, indicating the interface which should receive the request. This field is used to support devices with multiple interfaces.

wIndex, bits 15-8: are reserved.

wLength indicates the CBI Command Transport Data size in bytes.

Note that the value of *wLength* depends on the Subclass of the mass storage device, as encoded in the *bInterfaceSubClass* field of the Interface descriptor.

The host shall set *wLength* to the number of bytes of Command Transport Data to be transported, and transport exactly that number of bytes. If the value of wLength is not equal to the value expected by the device, the device shall fail the Command Transport, by stalling the ADSC. See section 2.3.1.

CBI mass storage devices will only receive one command block at a time. The host is required to order and queue requests to serialize their delivery to the device.